

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Strojnícka fakulta

Evidenčné číslo: SjF-13432-81384

Experimentálne moduly pre výučbu automatizácie

Bakalárska práca

2018

Tibor Konkoly

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Strojnícka fakulta

Evidenčné číslo: SjF-13432-81384

**Experimentálne moduly pre výučbu
automatizácie**

Bakalárska práca

Študijný program: automatizácia a informatizácia strojov a procesov

Študijný odbor: 5.2.14. automatizácia

Školiace pracovisko: Ústav automatizácie, merania a aplikovanej informatiky

Vedúci záverečnej práce: doc. Ing. Gergely Takács, PhD.

Konzultant: Ing. Martin Gulan, PhD.

Bratislava 2018

Tibor Konkoly

•••• S T U
•••• S j F

ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Tibor Konkoly**
ID študenta: **81384**
Študijný program: **automatizácia a informatizácia strojov a procesov**
Študijný odbor: **5.2.14. automatizácia**
Vedúci práce: **doc. Ing. Gergely Takács, PhD.**
Konzultant: **Ing. Martin Gulan, PhD.**
Miesto vypracovania: **ÚAMAI SjF STU v Bratislave**

Názov práce: **Experimentálne moduly pre výučbu automatizácie**

Jazyk, v ktorom sa práca vypracuje: **slovenský jazyk**

Špecifikácia zadania:

Úlohou študenta je vytvoriť hardwarový a softwarový rámec na tvorbu experimentálnych modulov pre mikroradičovú prototypizačnú dosku Arduino (AVR). Konkrétnie, študent musí vytvoriť optickú experimentálnu stanicu a experimentálnu stanicu s jednosmerným motorom. V rámci bakalárskej práce študent musí

- navrhnuť experimentálne zariadenie, vybrať vhodné elektronické a mechanické komponenty, navrhnuť elektrické zapojenie a plošný spoj,
- vypočítať a dôkladne testovať kvalitu a funkčnosť 10 ks. zariadení,
- napísať programátorské rozhranie (application programmers interface, API) pre ovládanie zariadenia,
- vytvoriť inštrukčné príklady vhodné na didaktické nasadenie, napríklad identifikáciu sústavy alebo spätnoväzobné riadenie, pritom príklady musia názorne predvíeť funkčnosť zariadenia,
- dôkladne zdokumentovať hardvér, softvér a príklady pomocou webstránky (Wiki) a podrobne komentovať zdrojový kód.

Rozsah práce: **cca. 30-40 strán**

Riešenie zadania práce od: **12. 02. 2018**

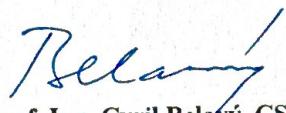
Dátum odovzdania práce: **28. 05. 2018**

Tibor Konkoly

študent


prof. Ing. Cyril Belavý, CSc.
vedúci pracoviska

Slovenská technická univerzita
v Bratislave
Dekanát Strojnickej fakulty
Útvor pedagogických činností
812 31 Bratislava, Nám. Slobody 17
-3-


prof. Ing. Cyril Belavý, CSc.
garant študijného programu

Čestné prehlásenie

Ja, Tibor Konkoly vyhlasujem, že záverečnú prácu som vypracoval samostatne, pod vedením vedúceho práce a s použitím uvedenej literatúry.

Bratislava, 20. mája 2018

.....
Vlastnoručný podpis

Pod'akovanie

Ďakujem vedúcemu záverečnej práce, doc. Ing. Gergelymu Takácsovi, PhD., za cenné rady a odbornú pomoc pri vypracovaní záverečnej práce. Ďakujem spolužiakom, Samuelovi Mladému a Jakubovi Mihálikovi za ich čas a pomoc pri nájdení a oprave chýb. Ďalej chcem pod'akovať Gáborovi Penzingerovi za nápady a pripomienky pri vypracovaní záverečnej práce.

Bratislava, 20. mája 2018

.....
Vlastnoručný podpis

Názov práce: Experimentálne moduly pre výučbu automatizácie

Kľúčové slová: Arduino UNO, AutomationShield, PID, Optoshield, Motoshield

Abstrakt: Cieľom bakalárskej práce je návrh experimentálnych modulov založených na platforme Arduino vo forme externých shieldov, ktoré sú určené na použitie vo výučbe. Skladajú sa z hardwareovej a softwareovej časti. V rámci bakalárskej práce boli navrhnuté dva moduly, Optoshield a Motoshield. Práca obsahuje popis tvorby spomínaných modulov. Začína predstavením použitých súčiastok a charakterizuje všetky celky elektronického obvodu, aby čitateľ pochopil na akom princípe funguje elektronický obvod. Pokračuje s návrhom a tvorbou schémy v programe DipTrace. V softwareovej časti sú popísané funkcie ktoré boli napísané pre jednotlivé moduly. Popis obsahuje syntax a princíp, na ktorom funkcia pracuje. Zdrojový kód funkcií sa nachádza pod popisom. Na konci sú vypracované didaktické príklady, v ktorých sú použité navrhnuté moduly. Sú popísané použité metódy, ciele príkladu a program je tiež vysvetlený. Celý zdrojový kód sa nachádza v prílohoch. Na konci príkladov sú graficky znázornené výsledky v programe Arduino IDE (z pohľadu užívateľa) a v programe MATLAB.

Title: Experimental modules for Automation Education

Keywords: Arduino UNO, AutomationShield, PID, Optoshield, Motoshield

Abstract: The main aim of the bachelor's thesis is to develop experimental modules based on Arduino for educational purposes. Each module consists of two parts - hardware and software. During the project two modules were designed, the Optoshield and the Motoshield. The thesis depicts the processes from design to experiments. Each module has its own chapter. Hardware part of the chapter starts with the description of used components. Knowing these components is important, if the user wants to know the working principles of the circuits used. The chapter continues with the schematic and printed circuit board design using DipTrace. The software part of chapters introduces the functions, which were written for making the work with the modules easier. It contains the description of the functions, working principle and also contains the source code of the function. Final chapter is about experiments, which use the introduced boards. Each experiment processes a principle from control engineering theory, like PID control or step response. Experiments include the description of the used principle and code, its goals and graphs, which represent the results from the view of the user (in Arduino IDE) and in called MATLAB. The source code of the experiments is included in the appendices.

Obsah

1	Úvod	1
2	Motivácia	2
3	Optoshield	6
3.1	Hardware	6
3.1.1	Popis súčiastok	6
3.1.2	Kreslenie schémy zapojenia	13
3.1.3	Vytvorenie plošného spoja	15
3.2	Software	18
3.2.1	Popis funkcií	19
4	Motoshield	27
4.1	Hardware	27
4.1.1	Popis súčiastok	27
4.1.2	Kreslenie schémy zapojenia a vytvorenie plošného spoja	33
4.2	Software	34
5	Didaktické príklady	40
5.1	Regulácia a PID regulátor	40
5.2	Skoková funkcia pomocou Optoshieldu	41
5.3	PID regulácia pomocou Optoshieldu	44
5.4	Identifikácia Optoshieldu	46
5.5	Skoková funkcia pomocou Motoshieldu	50
5.6	PID regulácia pomocou Motoshieldu	52
5.7	Identifikácia Motoshieldu	55
6	Záver	59
A	Zdrojový kód funkcie Sampling()	i
B	Skoková funkcia pomocou Optoshieldu	iv
C	PID regulácia pomocou Optoshieldu	vi
D	Skoková funkcia pomocou Motoshieldu	viii

E PID regulácia pomocou Motoshieldu	x
F Program na meranie údajov na identifikáciu Motoshieldu	xii

Zoznam obrázkov

2.1	Učebná pomôcka od firmy Quanser [16]	2
2.2	Prototypizačná doska Arduino	4
2.3	Arduino s pripojeným Motoshieldom	5
3.1	Schematická značka svetelnej diódy	6
3.2	Učebná pomôcka OptoShield v častiach	7
3.3	Spôsob zapojenia fotorezistora	8
3.4	OptoShield Schéma	8
3.5	Odporový delič napäťia - schéma zapojenia	9
3.6	Testovanie Optoshieldu s pôvodnými diódami	9
3.7	Výsledok testu s novou hodnotou R_3 a R_4 (1)	12
3.8	Výsledok testu s novou hodnotou R_3 a R_4 (2)	12
3.9	DipTrace Pattern Editor	13
3.10	DipTrace Component Editor	14
3.11	DipTrace Component Editor - Priradzovanie pinov	14
3.12	Plošný spoj OptoShield - predná strana	16
3.13	Plošný spoj OptoShield - zadná strana	16
3.14	Vyplnené údaje pri kontrole DFM	17
3.15	Učebná pomôcka OptoShield	17
4.1	Integrovaný obvod L293D s pomemovanými vývodmi	29
4.2	Schematická značka operačného zosilňovača	29
4.3	Operačný zosilňovač LM358	30
4.4	Meranie prúdu	30
4.5	Zapojenie operačného zosilňovača na odčítanie hodnôt napäťia	31
4.6	Zapojenie neinvertujúceho operačného zosilňovača	32
4.7	Hotová schéma MotoShield	33
4.8	Plošný spoj MotoShield - predná strana	33
4.9	Plošný spoj MotoShield - zadná strana	34
4.10	Hotový Motoshield	34
5.1	Regulačný obvod	40
5.2	Skoková funkcia	42
5.3	Odozva Optoshieldu na skokovú funkciu, Setpoint = 50	43
5.4	Odozva Optoshieldu na skokovú funkciu, Setpoint = 100	43
5.5	Odozva Optoshieldu na skokovú funkciu, Setpoint = 100	43
5.6	Grafický výstup z PID 1	45

5.7	Grafický výstup z PID 2	46
5.8	Priebeh PID regulácie zobrazený v programe MATLAB	46
5.9	Identification toolbox v programe MATLAB	47
5.10	Process Model s vygenerovanými konštantami	48
5.11	Zobrazené merané a simulované dátá	48
5.12	Zostavený model v programe Simulink	49
5.13	Výsledok simulácie	50
5.14	Aplikácia zistených konštánt	50
5.15	Aplikácia zistených konštánt	51
5.16	Skoková odozva Motoshieldu, Setpoint = 80	52
5.17	Skoková odozva Motoshieldu v programe MATLAB, Setpoint = 100	52
5.18	Grafický výstup z PID 1	54
5.19	Priebeh PID zobrazený v programe MATLAB	54
5.20	Pokrytie meraných dát modelom	56
5.21	Výsledok simulácie	57
5.22	Aplikované parametre regulátora	58

Zoznam tabuliek

3.1	Výber odporov R_1 a R_2 do deličky	10
3.2	Výber odporov R_3 a R_4 pre svetelné diódy	11
4.1	Pravdivostná tabuľka integrovaného obvodu L293D	29
4.2	Testovanie zapojenia na odčítanie hodnôt 1	31
4.3	Testovanie zapojenie na odčítanie hodnôt	32

1 Úvod

Cieľom bakalárskej práce je návrh a výroba moderných učebných pomôcok na výuku automatizačnej techniky.

Učebné pomôcky zohrajú dôležitú úlohu vo výučbe, ktorú robia zaujímavejším a atraktívnejším. Umožňujú ľahšie pochopenie učiva a zároveň môžu robiť proces učenia sa zábavou. Existujú mnohé firmy, ktorých hlavný profil je výroba a predaj špičkových moderných učebných pomôcok. Bohužiaľ, ceny týchto výrobkov sú vysoké a nie všetky školy a univerzity si môžu dovoliť ich kúpu. Z toho vyplýva, že školy a univerzity nie vždy disponujú s potrebnými učebnými pomôckami v dostatočnom množstve. Na tento problém ponúka riešenie projekt ústavu Automatizácie, merania a aplikovanej informatiky Strojníckej fakulty Slovenskej technickej univerzity v Bratislave.

V rámci projektu študenti sa zaoberajú s návrhom a výrobou učebných pomôcok vo forme externých rozšírení (ďalej shield) pre prototypizačné dosky Arduino UNO. Arduino ponúka moderné a finančne dostupné riešenie. Výhodou Arduina je, že existuje veľké množstvo pomocných materiálov vo forme kníh a článkov na internete, ktoré uľahčia vyriešenie prípadných problémov. Aj vďaka tomuto je odporúčaný pre začiatočníkov. Umožňuje pochopenie a osvojenie si základov elektroniky a programovania.

V bakalárskej práci je popísaný proces návrhu a výroby dvoch modulov, Optoshieldu a Motoshieldu. Každý modul sa skladá z hardwareovej a softwareovej časti. Na začiatku sú predstavené súčiastky použité na danom shielde. Užívateľ musí pochopiť na akom princípe daný shield funguje a preto predstavenie súčiastok je dôležité. Nasleduje proces návrhu schémy a plošného spoja v počítačovom programe DipTrace.

V softwareovej časti sú predstavené charakteristické funkcie daného shieldu. Je popísaný princíp, na ktorom je funkcia založená. Čitateľ sa dozvie akého typu je funkcia, akého typu sú hodnoty vrátené funkciou a ako sa funkcia správne používa. Po charakteristike je vložený zdrojový kód celej funkcie. Preddefinované funkcie umožňujú ľahšiu a efektívnejšiu prácu so shieldom.

Na konci každej časti sú pripravené príklady v ktorých je aplikovaný jeden princíp z oblasti automatizácie. Pred samotným príkladom je teoretický popis aplikovaného princípu, ak by čitateľ nemal predchádzajúce znalosti z oblasti automatizácie. Ďalej sú popísané ciele príkladu a sú vysvetlené dôležité časti zdrojového kódu, aby čitateľ vedel ako a s použitím ktorých funkcií bol daný princíp aplikovaný. Celý zdrojový kód sa nachádza medzi prílohmi práce. Na konci sú graficky znázornené výsledky z pohľadu užívateľa v programe Arduino IDE a v počítačovom programe MATLAB. Grafy z Arduino IDE neumožňujú presnú analýzu dát, skôr slúžia na vizualizáciu výsledkov, ale priloženie už spomínaných grafov považujem za dôležité, aby užívateľ mal spätnú väzbu a aby vedel, že či program funguje správne.

2 Motivácia

AutomationShield je projekt na Ústave automatizácie, merania a aplikovanej informatiky Slovenskej technickej univerzity v Bratislave. Projekt je zameraný na tvorbu učebných pomôcok vo forme externých rozšírení (tzv. shieldov) kompatibilných s Arduino UNO na výuku automatizačnej techniky. Hlavnou motiváciou projektu je malý počet finančne dostupných moderných učebných pomôcok na výuku automatizácie.

Učebné pomôcky umožňujú aplikovať na prednáškach prebraté metódy v praxi. Po- zostávajú sa z hardvérovej a softvérovej časti. Hardvér obsahuje hlavné komponenty ako senzory, akčné členy a ďalšie, ktoré sú nevyhnutné pre dosiahnutie funkčnosti. Softvérová časť obsahuje preddefinované funkcie napr. na čítanie senzorov, ovládanie akčných členov, časovanie. Skutočnú hodnotu produktu tvorí softvérová časť. V súčasnosti existujú firmy, ktoré sa zaoberejú výrobou a predajom rôznych učebných pomôcok, aj na výuku automatizačnej techniky, ale ich cena je vysoká.



Obr. 2.1: Učebná pomôcka od firmy Quanser [16]

Začali sme s hľadaním vhodnej hardvérovej platformy. Hlavné kritériá boli dostupnosť produktov, vhodnosť na naše účely ale aj cena. Vybrali sme si platformu Arduino.

Arduino je open-source platforma. Open-source znamená, že návrhy dosiek nie sú chránené patentom a sú voľne dostupné – každý môže mať svoju vlastnú verziu Arduina. Je to rodina prototypizačných dosiek, ktoré sa líšia od seba nielen veľkosťou ale aj výkonom.

Nápad pochádza z Talianska a pôvodne bol určený univerzitným študentom na osvojenie si základov elektroniky a programovania. Dneska sú dosky obľúbené nielen v kruhoch študentov ale aj u hobbystov. Vytvorila sa silná komunita, ktorej členovia zdieľajú na internete svoje návrhy a projekty, vznikali rôzne stránky a fóra, kde si užívatelia môžu vymieňať svoje skúsenosti, prípadne poradiť niekomu, kto má problém so svojim hardvérom alebo softvérom. Okrem internetových zdrojov sú dostupné aj knihy (prevažne v anglickom jazyku), ktoré môžu dobre slúžiť hlavne pre začiatočníkov. Užívateľ sa na začiatku zoznámi so základnými princípmi elektroniky, napíše svoj prvý kód a postupne sa naučí nové príkazy, zoznámi sa s novými súčiastkami. Hlavné výhody Arduina spočívajú v jednoduchosti, všestrannosti, cene a rôznorodosti. Na internete si nájdeme snímače všetkých druhov, ktoré sú kompatibilné, moduly na rôzne formy bezdrôtovej komunikácie a ďalšie súčiastky, ktoré si môžu nájsť uplatnenie v našich projektoch. Pomocou spomínaných súčiastok Arduino dokáže monitorovať svoje okolie a spracovávať získané dátu zo snímačov. Srdcom každej dosky Arduino je mikroradič od spoločnosti Atmel. Získané dátu posiela pre počítač použitím sériovej komunikácie. Okrem sériovej komunikácie Arduino dokáže aplikovať ďalšie druhy komunikácie¹.

V rámci predmetu mikroprocesorová technika sme sa zoznámili s učebnou pomôckou Flexi ktorá je založená na platforme Arduino. Nápad sa pochádza z ústavu informatizácie, automatizácie a matematiky FCHPT. Akčným členom je ventilátor ktorý zdvihne flex senzor do určitej výšky. Podľa toho, do akej miery je flex senzor ohnutý sa mení jeho odpór. Ďalej obsahuje potenciometer ako referenciu. S Flexim sme začali robiť na začiatku semestra a bol dobrou pomôckou. Začali sme so základmi, čítaním analógových údajov z potenciometra, merali sme hodnoty napäťia s multimetrom a zobrazili sme rôzne priebehy napäťia na osciloskope. Potom, ako základy boli prebraté, pokračovali sme s riešením komplexnejších úloh. Postupne sme sa zoznámili s rôznymi príkazmi Arduina, naučili sme sa ako sa používajú funkcie, cykly a ďalšie stavebné jednotky programov. Ku koncu semestra sme riešili prechodovú charakteristiku a pokročilú metódu časovania pomocou Timer registrov. Podobnou cestou som sa chcel vydať aj ja.

Pre naše účely sme si vybrali model Arduino UNO. Rozmery dosky sú 68 x 53 mm, obsahuje štrnásť digitálnych pinov a šesť analógových. Mikroradič je typu ATmega 328. Niektoré z digitálnych pinov (sú označené znakom ~) dokážu generovať PWM² signál, ktorý sa používa na nastavenie rýchlosťi motorov ale aj jasu svetelných diód.

Pre UNO existujú rozšírenia, takzvané shieldy. Shieldy sú plošné spoje plniace určité funkcie. Ich primárnym účelom je rozšírenie schopností Arduina, ich rozmery sa zhodujú s rozmermi Arduina. Shieldy s Arduinom môžeme spojiť pomocou špeciálnych predĺžených pinov. Dosku so shieldom spojíme vložením pinov shieldu do pinov dosky Arduino.

Podstata nášho projektu je integrácia konceptov do jedného shieldu. Je to výhodné riešenie z hľadiska modularity. Môžeme si pripraviť rôzne shieldy s rôznymi akčnými členmi a senzormi. Ku každému shieldu bude napísaná knižnica, ktorá obsahuje všeobecné funkcie platné pre celý projekt AutomationShield ale aj špecifické funkcie platné pre daný shield.

Projekt má vlastnú web stránku. Na úvodnej stránke je krátky popis projektu. Pod ním je zoznam aktuálnych shieldov. V zozname sa nachádzajú aj shieldy, ktoré sú vo fáze vývoja. Nižšie čitateľ nájde informácie o možnostiach výroby jednotlivých shieldov a

¹I2C, 1Wire, SPI.

²Pulse Width Modulation



Obr. 2.2: Prototypizačná doska Arduino

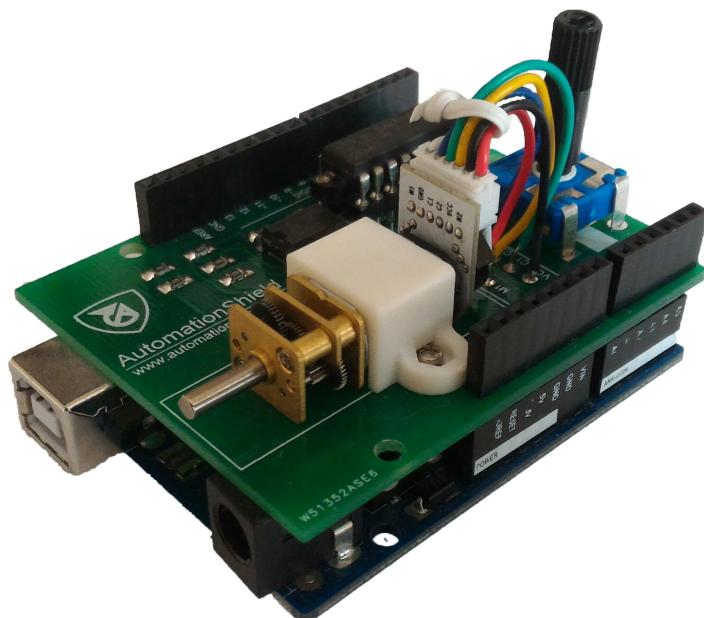
všeobecné informácie o knižnicách napísaných pre shieldy. V rámci zoznamu sú popísané jednotlivé shieldy. Čitateľ si nájde podrobný popis hardvéru a môže sa zoznámiť so súčasťami a ich funkciami, pričom pochopí princípy aplikované na danom shielde. Nasleduje popis softvérovej časti, ktorá obsahuje zoznam preddefinovaných špeciálnych funkcií pre daný hardvér. Každá funkcia je stručne charakterizovaná. V popise sa nachádza aj syntax a príklad na volanie funkcie. Súčasťou stránky sú pripravené príklady v ktorých sa používajú vopred popísané funkcie. Príklady umožňujú čitateľovi pochopiť súvislosti medzi jednotlivými funkciami. Po úspešnom absolvovaní pripravených príkladov čitateľ efektívnejšie vie použiť pomôcky. Stránka ďalej obsahuje fotky o shieldovi, shému zapojenia a hotový plošný spoj. Súbory schémy a plošného spoja sú stiahnutelné a umožňujú domácom výrobom alebo modifikáciu shieldov. Stránka každého shieldu obsahuje tabuľky, v ktorých je typ, počet a popis použitých súčiastok. Tabuľky pomáhajú pri cenových kalkuláciách.

Veľká výhoda projektu je, že nepotrebujeme špeciálny výstroj. Stačí, ak vlastníme dosku Arduino UNO a USB kábel na pripojenie k počítaču. Vzhľadom na to, že projekt je robený na škole, vieme si prispôsobiť jednotlivé shieldy pre naše potreby. Jedna z nevýhôd je, že sme pri navrhovaní shieldov obmedzený priestorom. Môžeme si ušetriť určité miesto použitím vhodných súčiastok, čo vysvetlím neskôr.

Do projektu sa môžu zapojiť študenti, môžu si navrhnúť ich vlastný a jedinečný shield, aj v rámci bakalárskej alebo diplomovej práce. Študenti ďalej môžu riešiť zložitejšie shieldy ako skupinové projekty v rámci jednotlivých predmetov. Benefity pre študentov sú jednoznačné. Aplikujú teoretické poznatky z elektroniky v praxi pri navrhovaní hardvérovej časti. Pochopia princípy fungovania použitých súčiastok a tieto poznatky im môžu byť užitočné v budúcnosti. Pri písaní softvérovej časti sa naučia základy objektovo orientovaného programovania a písania štrukturovaného kódu. Ak chceme aplikovať nejaký

princíp v programe ako funkciu, najprv musíme pochopiť problematiku, čo umožňuje aj zlepšenie poznatkov z oblasti automatizácie. Ďalej sa zlepšujú ich komunikačné zručnosti (aj v cudzom jazyku) a naučia sa robiť v tíme a ďalšie soft skilly, ktoré ich môžu robiť atraktívnejším na pracovnom trhu.

Vzhľadom na to, že softvérovú časť píšu viacerí programátori, spolupracujú a komunikujú na stránke GitHub. GitHub umožňuje programátorom z celého sveta spolupracovať a spoločne vyvíjať kód. Každý môže robiť samostatne svoju verziu a tie sú na konci zlúčené. Každý, kto je členom komunity GitHub vie editovať alebo okomentovať náš kód, čo prispieva k efektívnosti a ľahšiemu vyriešeniu prípadných problémov.



Obr. 2.3: Arduino s pripojeným Motoshieldom

3 Optoshield

3.1 Hardware

3.1.1 Popis súčiastok

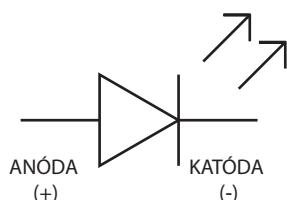
Optoshield je prvý zo série plošných spojov rodiny AutomationShield. V rámci Optoshieldu boli použité jednoduché a bežne dostupné súčiastky a výsledkom je plošný spoj, ktorý je lacný, jednoducho vyrobiteľný a umožňuje simuláciu základných princípov automatizácie.

Akčný člen je 5 mm biela LED dióda, ktorá zároveň slúži ako vstup do systému, $u(t)$. LED dióda (svetelná dióda alebo Luminiscenčná dióda) je polovodičová súčiastka, ktorá sa vyrába z kremíka. Skratka LED predstavuje Light Emitting Diode. Je dostupná v rôznych farbách a veľkostach. Typické farby sú červená, biela, modrá, žltá a zelená. Typické veľkosti sú 1.8 mm, 3 mm, 5 mm a 10 mm. Veľkosti predstavujú priemer súčiastky. Súčiastka má polaritu, kladná strana sa volá anóda a záporná strana sa volá katóda. Kladný vývod súčiastky je dlhší. Ak priviedieme napätie na kladný vývod svetelnej diódy a záporný vývod je uzemnený, tak svieti. Existujú aj v SMD podobe. Výhodou SMD súčiastok je úspora miesta.

Nesmieme zabudnúť na to, že pred svetelnou diódou musíme používať ochranný rezistor, ktorý chráni diódu pred poškodením. Veľkosť rezistora závisí od parametrov svetelnej diódy. Dôležité parametre sú pracovné napätie a prúdový odber svetelnej diódy. Tieto parametre si nájdeme väčšinou v katalógu výrobcu alebo v datasheete súčiastky. Veľkosť potrebného rezistora si vypočítame podľa Rov. 3.1 na strane 6.

$$R = \frac{V - V_{LED}}{I_{LED}} \quad [\Omega]. \quad (3.1)$$

V je pracovné napätie Arduina, a je to 5 V. V_{LED} je pracovné napätie svetelnej diódy, kým I_{LED} je prúdový odber svetelnej diódy.

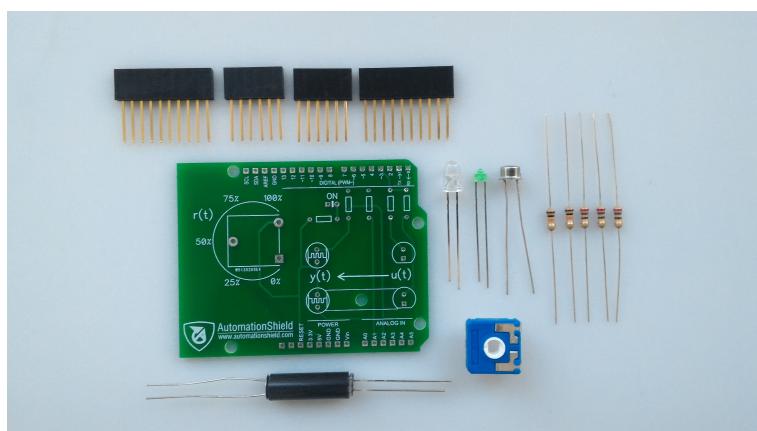


Obr. 3.1: Schematická značka svetelnej diódy

Ako referencia $r(t)$ je použitý potenciometer. Potenciometer je súčiastka s reguľovateľnou hodnotou odporu. Dva vývody potenciometra sú zapojené na napätie a zem. Tretí, takzvaný bežec, je pripojený na analógový pin, ktorý umožňuje čítanie hodnoty potenciometra. Vzhľadom na to, že analógovo-číslicový prevodník Arduina je desať bitový, prevodníkom vrátené hodnoty sú od 0 do 1023, pričom 1023 je najväčšia hodnota. Potenciometre majú nominálny odpor, potenciometer aplikovaný na Optoshield má nominálny odpor $10\text{ k}\Omega$. Nominálny odpor vieme zmerať multimeterom. Zapneme si multimeter a zvolíme si meranie odporu a vývody priložíme k dvom vývodom potenciometra (napätie a zem). Vzhľadom na to, že v automatizácii pracujeme prevažne s percentami, na plošnom spoji je stupnica pre hodnoty potenciometra v percentách, podľa ktorej si užívateľ vie nastaviť požadovanú referenčnú hodnotu. Stupnica potenciometra je zobrazená na Obr. 3.2.

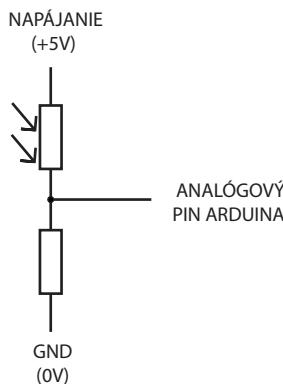
Svetelné diódy sú pripojené k digitálnemu pinu D3, ktorý je schopný generovať PWM signál. Skratka PWM znamená Pulse Width Modulation, čiže regulujeme šírku výstupného signálu. V prípade Arduina slúži na reguláciu výstupného napäcia. Vzhľadom na to, že je to digitálny signál, pracuje v rozsahu 0-255. 100% predstavuje hodnotu 255 a 5 V, hodnota 75% predstavuje hodnotu 191 a 3.75 V, 50% predstavuje hodnotu 127 a 2.5 V, 25% predstavuje hodnotu 64 a 1.25 V. Pomocou PWM signálu a potenciometra si vieme nastaviť jas svetelných diód.

Výstupom zo systému $y(t)$ je senzor na meranie jasu. Na Optoshield bol použitý fotorezistor. Fotorezistor je pasívna polovodičová súčiastka, ktorej odpor je nepriamo úmerný veľkosti jasu. To znamená, že čím nižšia je hodnota jasu tým väčší odpor má fotorezistor. Fotorezistor má nominálny odpor. Na Optoshield sú $10\text{ k}\Omega$ fotorezistory, ktoré tvoria páry so svetelnou diódou. Prvý párs je umiestnený v uhlíkovej modelárskej trubici. Uhlíková trubica eliminuje vonkajšie rušivé svetlo a zvýši presnosť vykonaných meraní. Pri príkladoch a simulácii budeme používať prvý párs uložený v trubici. Druhý párs je voľne umiestnený vedľa prvého a slúži na experimentovanie a na vyskúšanie základných funkcií začiatočníkmi. Typické zapojenie fotorezistora je na Obr. 3.3 ktorý je na strane 8. Na Obr. 3.2 môžete vidieť páry tvorené fotorezistorom a svetelnej dióde a miesto ich uloženia na plošnom spoji.



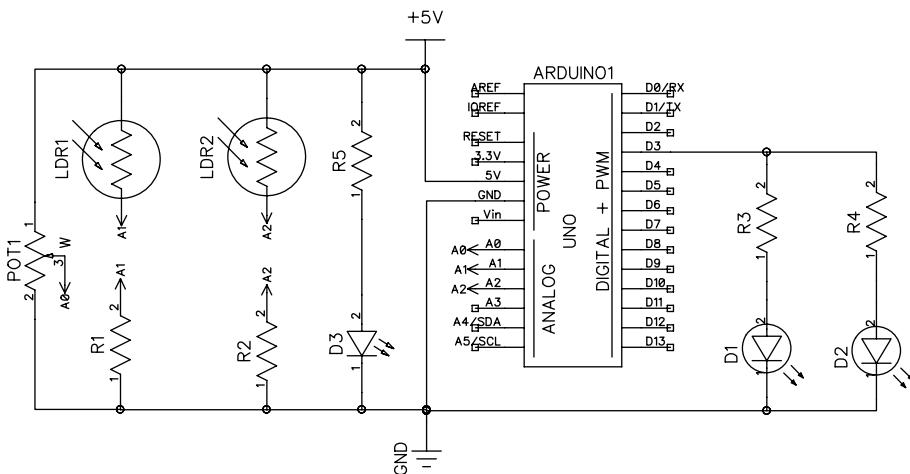
Obr. 3.2: Učebná pomôcka OptoShield v častiach

Na Obr. 3.3 vidíme, že fotorezistor je zapojený s ďalším rezistorom a spolu tvoria odporový delič napäcia. Odporový delič je jednoduchá schéma, ktorá je široko používaná v



Obr. 3.3: Spôsob zapojenia fotorezistora

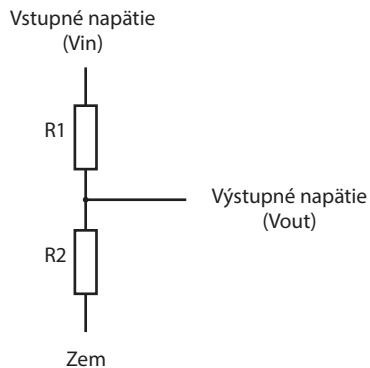
elektronike. Skladá sa z dvoch rezistorov a slúži na zmenšenie hodnoty vstupného napäťia. Z toho vyplýva, že výstupné napätie je úmerné vstupnému napätiu. Konštantu úmernosti si nastavíme pomocou hodnôt použitých rezistorov R_1 a R_2 . Všeobecnú schému odporového deliča môžete vidieť na Obr. 3.5. Odporový delič v schéme zapojenia môžete vidieť na Obr. 3.4 ktorý je na strane 8, kde fotorezistor je zapojený namiesto R_1 . Pomocou Rov. 3.2 si môžeme vypočítať výstupné napätie z odporového deliča.



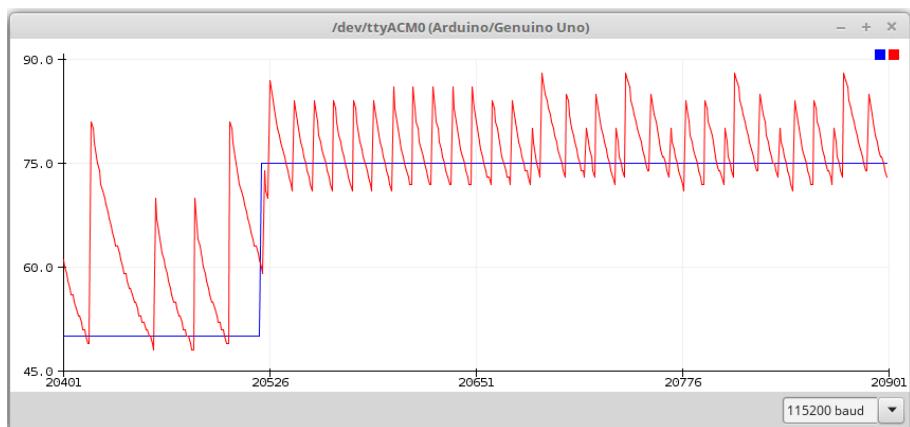
Obr. 3.4: OptoShield Schéma

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2} \quad [V]. \quad (3.2)$$

Primárne rezistory R_1 a R_2 mali hodnotu $10\text{ k}\Omega$, čiže identické s nominálnym odporom fotorezistora. Pred svetelnými diódami boli aplikované rezistory R_3 a R_4 s hodnotou $270\text{ }\Omega$, viď. Obr. 3.4 na strane 8. Prvé testy ale ukázali, že hodnoty rezistorov sú nevhodné. Jas svetelných diód bol príliš ostrý a hodnoty zo senzora boli veľmi podobné. Jas svetelných diód bol príliš ostrý a hodnoty zo senzora boli veľmi podobné. Na Obr. 3.6 môžeme vidieť test na skokovú odozvu. Modrou farbou je zobrazený jas svetelných diód a červenou farbou sú zobrazené analógové hodnoty z fotorezistora. Obidve hodnoty sú v %.



Obr. 3.5: Odporový delič napäťia - schéma zapojenia



Obr. 3.6: Testovanie Optoshieldu s pôvodnými diódami

Pri riešení problému prišli do úvahy dve možnosti – použitie svetelných diód inej farby alebo zvýšenie odporu rezistorov, čo má za následok stlmenie jasu. Ako prvé boli vymené rezistory. Odpor fotorezistora bol zmeraný multimeterom v dvoch prípadoch: keď svetelná dióda svetila naplno a keď svetelná dióda nesvetila. Keď svetelná dióda nesvetila tak fotorezistor mal odpor blízko nominálnej hodnoty. V prípade, keď svetelná dióda svetila naplno, odpor fotorezistora bol výrazne nižší, hodnota odporu bola 532Ω alebo $0.534 \text{ k}\Omega$. Po dosadení zmeraných hodnôt do Rov. 3.2 si vypočítame veľkosť výstupného napäťia z odporového deliča. Výpočty znázorňujú Rov. 3.3 - 3.5.

Výpočet výstupného napäťia z odporového deliča so starými hodnotami:

$$V_{outMin} = 5 \text{ V} * \frac{10 \text{ k}\Omega}{10 \text{ k}\Omega + 10 \text{ k}\Omega} = 2.5 \text{ V}, \quad (3.3)$$

$$V_{outMax} = 5 \text{ V} * \frac{10 \text{ k}\Omega}{10 \text{ k}\Omega + 0.534 \text{ k}\Omega} = 4.74 \text{ V}, \quad (3.4)$$

$$V_{out} = V_{outMax} - V_{outMin} = 4.74 \text{ V} - 2.54 \text{ V} = 2.24 \text{ V}. \quad (3.5)$$

Pomocou zmeraných hodnôt si vypočítame odpor ideálneho rezistora:

$$R_{2ideal} = \sqrt{R_{Max} * R_{Min}} = \sqrt{10 \text{ k}\Omega * 0.534 \text{ k}\Omega} = 2.3 \text{ k}\Omega. \quad (3.6)$$

Tabuľka 3.1: Výber odporov R_1 a R_2 do deličky

Jas (%)	Analógové hodnoty fotorezistora $R_1 = 10 \text{ k}\Omega$	Analógové hodnoty fotorezistora $R_1 = 2.4 \text{ k}\Omega$
100	948	768
95	944	764
90	941	760
85	938	756
80	937	755
75	937	747
70	935	738
65	927	724
60	924	710
55	920	703
50	917	699
45	911	693
40	901	688
35	888	671
30	865	633
25	857	587
20	837	541
15	815	470
10	756	405
5	689	311
0	*	50
Rozdiel	259	457

Z výpočtov vyplýva, že ideálny odpor je $2.3 \text{ k}\Omega$. Hodnoty rezistorov R_1 a R_2 boli zmenené na $2.4 \text{ k}\Omega$, lebo rezistor s odporom $2.3 \text{ k}\Omega$ neexistuje.

$$V_{outMin} = 5 \text{ V} * \frac{2.4 \text{ k}\Omega}{10 \text{ k}\Omega + 2.4 \text{ k}\Omega} = 0.97 \text{ V}, \quad (3.7)$$

$$V_{outMax} = 5 \text{ V} * \frac{2.4 \text{ k}\Omega}{0.534 \text{ k}\Omega + 2.4 \text{ k}\Omega} = 4.09 \text{ V}, \quad (3.8)$$

$$V_{out} = V_{outMax} - V_{outMin} = 4.09 \text{ V} - 0.97 \text{ V} = 3.12 \text{ V}. \quad (3.9)$$

Po výmene rezistorov nasledovalo overenie správnosti výpočtu. Použitá hodnota odporu bola dosadená do Rov. 3.2, viď. Rov. 3.7 - 3.9. Výsledné napätie je väčšie o 0.88 V ($3.12 \text{ V} - 2.24 \text{ V}$), čo môžeme vidieť aj pri značnej zmene rozlíšenia čo ukazujú tabuľkové hodnoty v Tab. 3.1 na strane 10.

Rezistory R_3 a R_4 nevyhovovali na reguláciu jasu svetelných diód, preto boli skúšané rôzne hodnoty odporov, ktoré vidíte v Tab. 3.2 na strane 11.

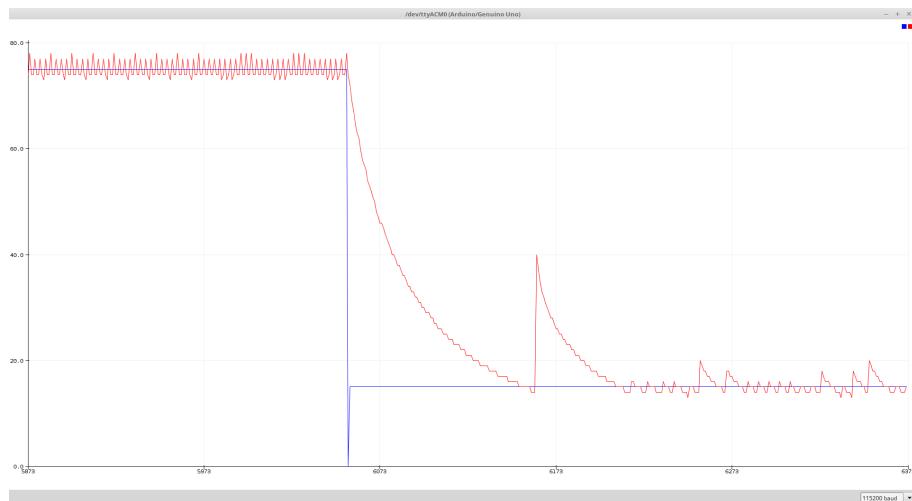
Počas merania bolo zistené, že čím väčší je odpor rezistorov R_3 a R_4 , tým nižšie rozlíšenie bude mať fotorezistor, ale zároveň hodnoty boli stabilnejšie pri jednotlivých úrovniach jasu. Na konci bol vybratý odpor $4.7 \text{ k}\Omega$. Pomocou testov bolo dokázané, že nová hodnota odporu je lepšia ako predchádzajúca, hodnoty z fotorezistora sú presnejšie

Tabuľka 3.2: Výber odporov R_3 a R_4 pre svetelné diódy

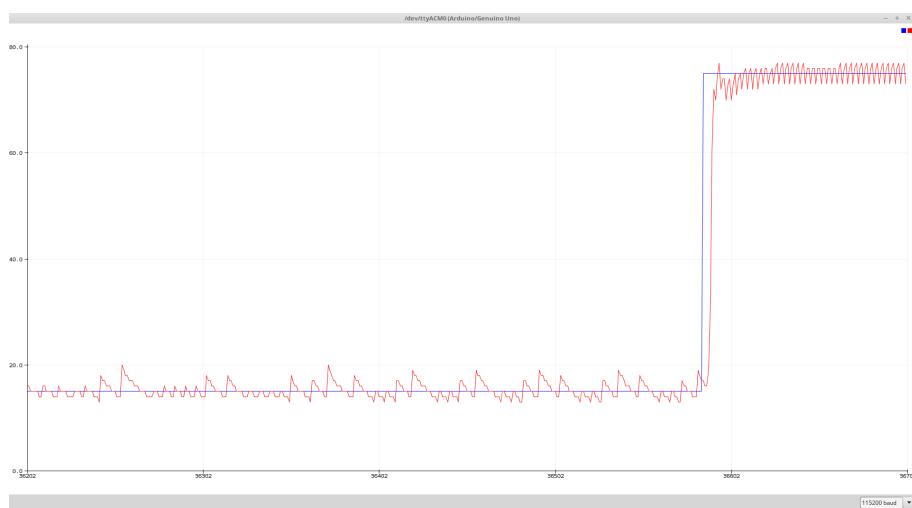
Jas (%)	Analógové hodnoty fotorezistora		
	$R_3 = 3.6 \text{ k}\Omega$	$R_3 = 4.7 \text{ k}\Omega$	$R_3 = 5.6 \text{ k}\Omega$
100	613	572	541
95	610	563	532
90	600	556	522
85	583	543	511
80	588	534	504
75	574	530	490
70	567	510	485
65	558	515	468
60	532	490	454
55	525	474	430
50	509	458	420
45	497	438	399
40	484	418	387
35	449	400	366
30	415	373	353
25	393	349	320
20	348	325	289
15	311	283	247
10	262	231	205
5	178	153	143
0	50	50	53
Rozdiel	435	419	398

a sú menšieho rozsahu. Výsledok testu si môžete vidieť na Obr. 3.7 a na Obr. 3.8. *

Poznámka k Tab. 3.2 - Hodnoty rezistorov R_3 a R_4 sú rovnaké.



Obr. 3.7: Výsledok testu s novou hodnotou R_3 a R_4 (1)



Obr. 3.8: Výsledok testu s novou hodnotou R_3 a R_4 (2)

Okrem predstavených súčiastok (fotorezistor, potenciometer a biela svetelná dióda) schéma obsahuje jednu 1.8 mm zelenú svetelnú diódu a jej ochranný rezistor R_5 . Táto dióda má jednoduchú funkciu, svieti ak plošný spoj je pod napäťom a zároveň signalizuje zapnutý stav.

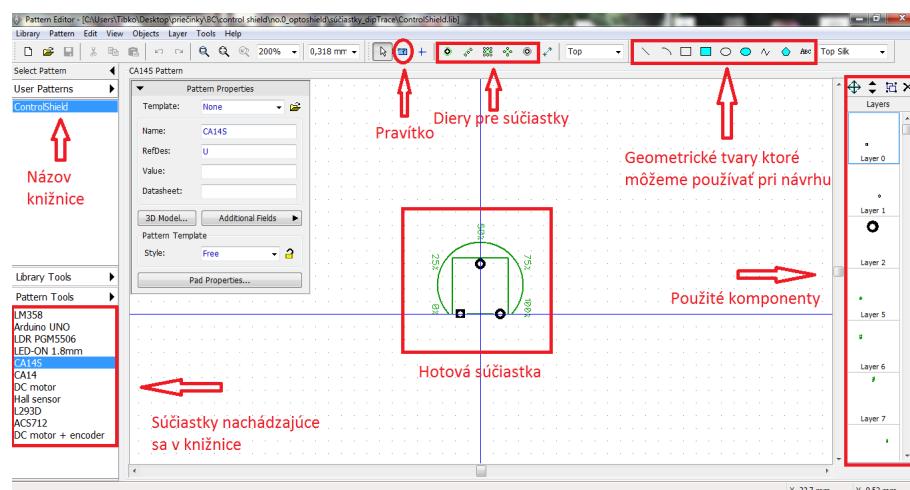
3.1.2 Kreslenie schémy zapojenia

Schéma bola nakreslená v bezplatnej verzii programu DipTrace. Bezplatná verzia má svoje obmedzenia, plošný spoj môže mať len dve strany a môžeme používať maximálne 300 pinov.

Začíname s hľadaním potrebných súčiastok. Samozrejme sa môže stať, že nie všetky potrebné súčiastky sa nachádzajú v databáze programu. V tomto prípade program ponúka možnosť, vďaka ktorej si ju môžeme vytvoriť. Na začiatku bola vytvorená knižnica s názvom ControlShield, ktorá obsahuje všetky vytvorené súčiastky v rámci projektu. Zároveň, knižnica uľahčí návrh budúcich plošných spojov.

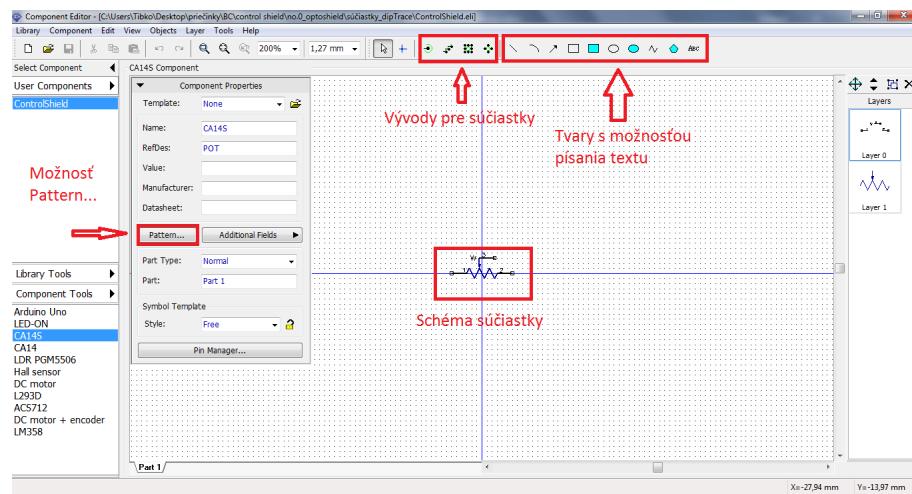
Vytvorenie novej súčiastky sa skladá z dvoch častí. V prvej časti si otvoríme DipTrace Pattern Editor v ktorej si nakreslíme vzor súčiastky, ako bude vyzerat na plošnom spoji. Tu nám pomôže katalóg alebo datasheet súčiastky, pretože presné rozmiestnenie dier pre jednotlivé vývody súčiastky je kľúčové aby nám súčiastka pasovala a aby bol plošný spoj funkčný. Môžeme si vytvoriť vzory s predvŕtanými dierami pre piny súčiastok alebo aj vzory pre SMD súčiastky, ktoré sa montujú na povrch plošného spoja. Z katalógu si určíme rozmer súčiastky a podľa tých rozmerov si rozmiestníme diery. V programe je klasický súradnicový systém X,Y, vďaka ktorému si vieme rozmiestniť naše súčiastky presným zadávaním súradníc. V programe v hornej lište sa nachádza znak pravítka, po kliknutí na toto pravítko si vieme zmerať vzdialenosť medzi vybranými bodmi – slúži na overenie presnosti rozmiestnenia dier. Jednotky si nastavíme na milimetre kliknutím na View → Units → mm.

Program nám ponúka ďalšie možnosti, ako kreslenie rôznych geometrických tvarov alebo priloženie textu a obrázku. Tieto nakreslené tvary alebo priložené texty a obrázky sa nachádzajú primárne vo vrstve TopSilk. Všetko, čo sa nachádza v tejto vrstve je zobrazené bielou farbou na povrchu plošného spoja. Kombináciou geometrických tvarov a textu som si vytvoril stupnicu pre potenciometer. Jednoduché geometrické tvary nám pomôžu napríklad pri označení prvého vývodu integrovaných obvodov alebo pri ohraničení priestoru pre súčiastky. Zároveň, použitie rôznych tvarov pre jednotlivé súčiastky robí plošný spoj estetickejším. Na Obr. 3.9 je pohľad na Pattern Editor s označenými vyššie predstavenými funkiami.



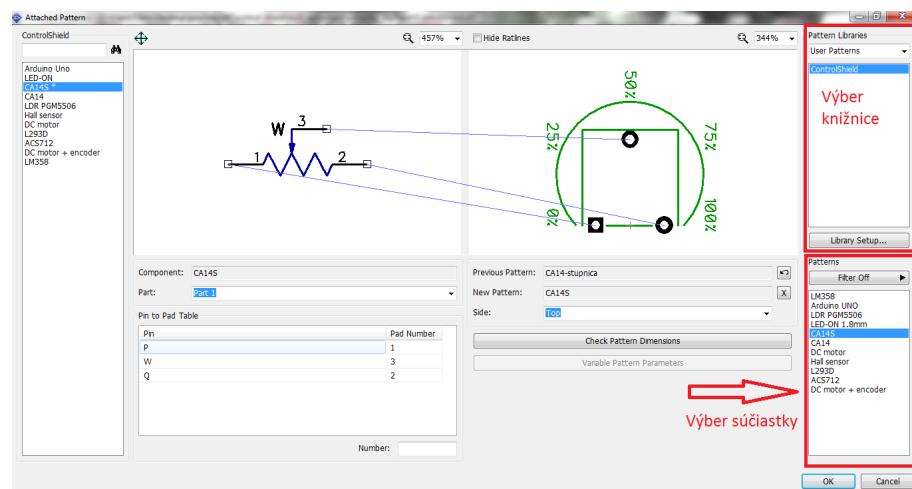
Obr. 3.9: DipTrace Pattern Editor

V druhej časti si otvoríme DipTrace Component Editor a nakreslíme si schému našej súčiastky. Pri výbere schémy by sme sa mali držať k pravidel platných v elektronike. Pri Optoshield som si nakreslil nami používaný potenciometer dvomi spôsobmi – so stupnicou a bez stupnice ale používal som všeobecnú schematickú značku platnú pre potenciometre pri obidvoch verziach. Platí, že schematická značka súčiastky nemusí splňať žiadne geometrické a rozmerové obmedzenia. Na Obr. 3.10 je pohľad na DipTrace Component Editor s nakreslenou schematickou značkou potencimetra.



Obr. 3.10: DipTrace Component Editor

Po nakreslení schematickej značky klikneme na "Pattern..." možnosť, kde si zvolíme správny, už nami vopred nakreslený vzor súčiastky a spojíme piny schémy s pinmi súčiastky. Takto dokončenú súčiastku si uložíme a je použiteľná pri návrhu ďalších plošných spojov. Proces je zobrazený na Obr. 3.11 na strane 14.



Obr. 3.11: DipTrace Component Editor - Priradovanie pinov

Po vytvorení chýbajúcich súčiastok sa môžeme pustiť do kresenia schémy. Ako prvé si umiestníme potrebné súčiastky v pracovnom prostredí programu. Súčiastky musia byť

umiestnené prehľadne, nechajme si dostatočný priestor medzi súčiastkami. Program ponúka možnosť rotácie súčiastok stlačením R alebo medzerníka. Ak sa všetky súčiastky nachádzajú v pracovnom prostredí, nasleduje pospájanie jednotlivých vývodov súčiastok. Robíme to kliknutím na "Place Wire", čo sa nachádza v hornej lište. Ak všetky potrebné vývody súčiastok sú spojené tak schému považujeme za hotovú. Prednastavená farba čiar je čierna. Farby čiar si môžeme zmeniť nasledovne: Pravý klik na čiaru → "Wire Color" → "Custom...". Pri kreslení schémy som použil pravidlo, že napájanie má červenú farbu, zem má modrú farbu, cesty vedúce z digitálnych pinov majú oranžovú farbu a cesty vedúce z analógových pinov majú tmavo zelenú farbu. Hotová schéma sa nachádza na Obr. 3.4 na strane 8.

3.1.3 Vytvorenie plošného spoja

Predtým, než si vytvoríme plošný spoj, musíme si overiť funkčnosť schémy stavbou prototypu. Prototyp bol robený na nepájivom kontaktnom poli. Hlavná výhoda nepájivých kontaktných polí je, že nemusíme spájkovať a súčiastky môžu byť použité viackrát. Po odstránení prípadných chýb schému môžeme považovať za správnu a môžeme si začať navrhovať plošný spoj.

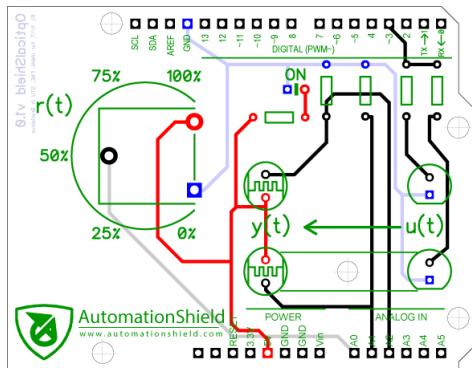
Pri návrhu plošného spoja vychádzame z nakreslenej schémy zapojenia. V súbore typu .dch, klikneme na "File" → "Convert to PCB". Súčiastky sú spojené modrými pomocnými čiarami. Tieto čiary nám pomáhajú pri spájaní súčiastok, označia čo s čím má byť spojené. Pomocné čiary niekedy môžu byť mätúce, lebo poradie súčiastok nie je vždy zodpovedné schémy.

Prvým krokom pri tvorbe plošného spoja je rozmiestnenie súčiastok. Správnym rozmiestnením súčiastok si uľahčíme tvorbu ciest. Plošný spoj po rozmiestnení súčiastok musí byť prehľadný, súčiastky si môžeme rozmiestniť podľa určenia. Ako môžete vidieť na Obr. 3.12 na strane 16, potenciometer ako referencia sa nachádza vľavo, rezistory a ON dióda boli umiestnené vpravo hore a akčné členy so senzormi, obidva páre sú nižšie pod rezistormi. Aj užívateľ sa vie ľahšie stotožniť so zapojením, ak súčiastky na plošnom spoji sú prehľadne umiestnené.

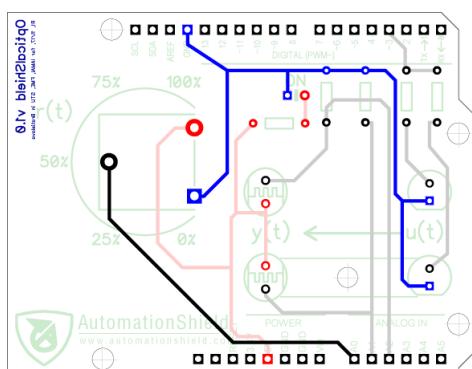
Program ponúka možnosť "Auto Placement" – automatické rozmiestnenie súčiastok. Ak sme začiatočníci, tak túto možnosť môžeme brať ako základ a postupnými zmenami si vieme plošný spoj prispôsobiť pre naše požiadavky. Po rozmiestnení súčiastok nasleduje vytvorenie ciest. Program ponúka možnosť "Auto Routing", ale neodporúčam jeho používanie. Plošný spoj robený autoroutingom je často nefunkčný lebo cesty sú skrížené. Súčiastky spojíme použitím tlačidla "Route Manual". Treba si uvedomiť nejaké pravidlá pri spájaní súčiastok. Program ponúka štandardnú šírku ciest, ktorú si vieme prispôsobiť našim požiadavkám. Cesty sa nemôžu krížiť, len v prípadoch keď to schéma vyžaduje. Spojenie nevhodných ciest môže poškodiť súčiastky a zároveň plošný spoj bude nefunkčný. Ďalej si musíme dávať pozor na tvar ciest. Musíme predchádzať kolmým spojeniam. Tvary ciest si môžeme zmeniť pomocou tlačidiel "Edit Traces" a "Free Trace Editing". Cesty môžemeťahať aj na hornej aj na spodnej strane plošného spoja. Aktuálne nastavenú stranu vidíme na hornej lište. Treba si vytvoriť vetvu pre napájanie a vetvu pre zem. Väčšinou somťahať jednu zo spomínaných vetiev na hornej strane plošného spoja a druhú vetvu na dolnej. Ak používame integrované obvody, môžemeťahať cesty medzi ich vývodmi, len treba si dávať pozor aby sme si ich nespojili. Môžeme si zmeniť farbu ciest pre lepsiú

prehľadnosť. Pre napájanie bola použitá červenú farbu a pre zem modrá. Ak sú súčiastky spojené, plošný spoj považujeme za hotový. Horná strana plošného spoja sa nachádza na Obr. 3.13 a spodná strana sa nachádza na Obr. 3.14.

Program ponúka možnosť kontroly v sekcií Verification – Check Design Rules. Ak program usúdi, že všetko je v poriadku, vypíše hlásenie "no errors found".



Obr. 3.12: Plošný spoj OptoShield - predná strana



Obr. 3.13: Plošný spoj OptoShield -zadná strana

Pre výrobu plošného spoja sú nevyhnutné súbory typu Gerber. Gerber je štandardný formát používaný na výrobu plošných spojov a je generovaný CAD programom. Klikneme na "File" – "Export" a program nám ponúka dva typy Gerber súborov: Gerber a Gerber X2. Odporúčam vygenerovať súbory typu Gerber X2, lebo tieto obsahujú aj súbory potrebné na vŕtanie montážnych dier. Súbory generujeme pre všetky vrstvy, je ich až pätnásť¹, osobitne. Odporúčam si ich uložiť do novej zložky. Pomocou rôznych zobrazovačov Gerber súborov, niektoré sú aj zadarmo, si vieme overiť vzhľad nášho plošného spoja nahraním programom vygenerovaných súborov.

Externá webstránka bola použitá na kontrolu plošných spojov, ktorá ponúka možnosť inštantnej DMF² kontroly. Táto kontrola je veľmi užitočná, upozorňuje nás na chyby v dizajne a vytvorí prvotnú vizualizáciu plošného spoja, zvlášť hornej a spodnej strany.

¹BoardOutline, Bottom, BottomAssy, BottomDimension, BottomMask, BottomPaste, BottomSilk, Drill_NonPlated_Through, DrillPlatedThrough, Top, TopAssy, TopDimension, TopMask, TopPaste, TopSilk

²Design for Manufacturability

Časté chyby, ktoré sa vyskytujú sú napríklad nedostatočná vzdialenosť medzi dierami a cestami alebo diery sú príliš blízko k okrajom plošného spoja. Na stránku si nahráme vygenerované Gerberove súbory združené do .rar alebo .zip súboru. Pomenujeme si plošný spoj a zadáme si naše meno a našu e-mailovú adresu. Vyberieme si farbu plošného spoja a hrúbku medenej vrstvy. Potom klikneme na "Submit", dĺžka kontroly závisí od zložitosti a počtu komponentov plošného spoja. Ak kontrola je hotová objaví sa report s možnými chybami, obrázkami ako vyzerá plošný spoj a ďalšími informáciami. Report zároveň posielajú aj na nami zadanú e-mailovú adresu v .pdf formáte. Ako [1] je uvedená adresa mnou použitej stránky.

Name
OptoShield

Email
(Remember, we're going to send you a link to your results once ready.)
adresa@email.com

Soldermask color
Green

Silkscreen color
White

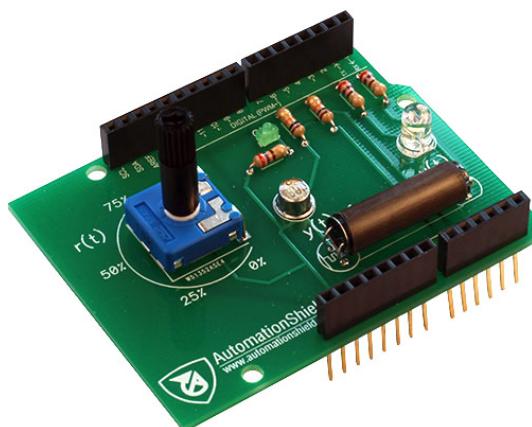
Thickness
0.062 in

File Upload
Compressed Gerber RS-274X, Gerber X2 or ODB++ files in .ZIP or .RAR format.
Vybrať súbor gerber.rar

SUBMIT

Obr. 3.14: Vyplnené údaje pri kontrole DFM

Návrh plošného spoja môžeme považovať za hotový. Ďalší krok je výroba. Existujú rôzne metódy na domácu výrobu plošných spojov, ale kvalita a estetická úroveň závisia od typu zvolenej metódy a od skúseností tvorcu. Pri niektorých metódach musíme pracovať s rôznymi kyselinami, ktoré sú nebezpečné na ľudské zdravie a na environment, zároveň si treba vyriešiť ich skladovanie. Vzhľadom na to, že plošné spoje budú použité vo výučbe, potrebovali sme viac kusov a preto boli vyrobené zákazkovo. Výsledok môžete vidieť na Obr. 3.13 na strane 16.



Obr. 3.15: Učebná pomôcka OptoShield

3.2 Software

Plošné spoje vyrobené v rámci projektu AutomationShield sa programujú jazykom Arduino, čo je zjednodušená verzia jazyku C. Napísaním jedného príkazu môžeme zavolať v pozadí ďalšie príkazy, s ktorými sa nemusíme zaoberať. Pôvodne bol určený študentom a ľuďom, ktorí nemali skúsenosti s programovaním. Výhody tohto spôsobu sú, že rýchlejšie prídeme na to ako jednotlivé príkazy fungujú, rýchlejšie si osvojíme základné štruktúry programov a dokážeme vygenerovať funkčné kódy. Hlavnou nevýhodou je fakt, že naše znalosti budú len povrchové a nepochopíme programovací jazyk až do úplnej hĺbky.

Pre celý projekt bola napísaná jedna knižnica, ktorej názov je AutomationShield. Knižnice pre Arduino projekty sú písané v jazyku C++ a sú použité princípy objektovo orientovaného programovania. Objektovo orientované programovanie spočíva v použití tried. Na písanie knižnice môžeme používať editory ako CodeBlocks, DevC++ ale aj Arduino IDE³.

Väčšinou knižnica sa skladá z dvoch súborov. Jeden je hlavičkový súbor, ktorý je typu .h a obsahuje triedy. Trieda sa skladá z dvoch sekcií, public a private. Do public, čiže verejnej časti sú napísané mená charakteristických funkcií použitých v rámci danej triedy a premenné ktoré sú dostupné aj pre externé funkcie. V sekcií private, čiže v súkromnej sa nachádzajú premenné ktoré sú použité v rámci funkcií triedy a nie sú dostupné pre externé funkcie. Zaradením premennej do privátnej sekcie vieme predchádzať nechcenú zmene hodnoty počas písania kódu.

Zdrojový kód 3.1: Štruktúra hlavičkového súboru

```
class OptoShieldClass {  
  
public:  
  
private:  
}; // koniec triedy
```

Na začiatok hlavičkového súboru dáme nasledujúce príkazy.

Zdrojový kód 3.2: Začiatok hlavičkového súboru

```
#ifndef AutomationShield_h // Zabranuje viacnasobne  
// vkladanie suboru  
#define AutomationShield_h // Ak nie je zadefinovany  
// zadefinuj  
  
#if (ARDUINO >= 100) // Overenie verzie softveru  
#include "Arduino.h" // Pre novu verziu Arduino IDE  
#else  
#include "WProgram.h" // Pre staru verziu Arduino IDE
```

³Integrated Development Environment

```
// Obsah suboru
```

```
#endif
```

Začiatočná podmienka zobrazená nižšie zabráni nechcenému viacnásobnému zadefinovaniu knižnice [6].

Zdrojový kód 3.3: Začiatočná podmienka

```
#ifndef AutomationShield_h
#define AutomationShield_h

//kod ide sem

#endif
```

Podmienka `#if (ARDUINO >= 100)` overuje verziu programu Arduino IDE. Ak je verzia novšia ako 1.00 tak príkazom `#include "Arduino.h"` vložíme všetky základné funkcie použité v jazyku Arduino, to isté urobí príkaz `#include "WProgram.h"` pre staršie verzie programu. Hlavičkový súbor končíme príkazom `#endif`, bez ktorého knižnica nefunguje.

Štruktúra knižnice AutomationShield je taká, že jedna trieda s názvom AutomationShield obsahuje všeobecné funkcie, ktoré môžu byť použité všetkými shieldmi. Takéto funkcie sú napríklad funkcia na PID reguláciu, funkcia na časovanie alebo veľmi užitočná funkcia `mapFloat()`, ktorá bude popísaná neskôr. Ďalej každý shield bude mať svoju vlastnú triedu pre osobitné funkcie charakterizujúce daný shield. Kvôli jednotnosti a prehľadnosti názvy niektorých funkcií môžu byť rovnaké pri rôznych shieldoch, hlavne keď ide o štandardné funkcie napr. na čítanie senzorov alebo referencie. Pre každú triedu si zadefinujeme externú instanciu s názvom triedy. Pre Optoshield to vyzerá nasledovne

`extern OptoClass OptoShield;` a plní funkciu takzvaného konštruktora. Konštruktor používame v .cpp súbore na asociáciu funkcií s danou triedou nasledovne:

```
OptoClass OptoShield;
```

Další súbor knižnice je súbor typu .cpp v ktorom sa nachádzajú všetky funkcie ktoré používame v rámci danej triedy. Na začiatku súboru príkazom

`#include "AutomationShield.h"` bol vložený už vopred napísaný hlavičkový súbor a pomocou príkazu `#include "Arduino.h"` boli vložené štandardné funkcie použité v jazyku Arduino.

3.2.1 Popis funkcií

V automatizácii fyzikálne veličiny sú vyjadrené prevažne v percentách. Veľakrát potrebujeme konvertovať hodnoty z rôznych rozsahov do rozsahu 0-100. Jazyk Arduino má jednu funkciu určenú na prevod hodnôt. Táto funkcia je

`map(premenná, min., max., nový min., nový max.).` Napíšeme si premennú s rozsahom v ktorom sa nachádzajú jej hodnoty a potom nový rozsah do ktorého si chceme

prekonvertovať staré hodnoty. Táto funkcia pracuje dobre, jediná jej nevýhoda, že pracuje s celými číslami typu *integer* – *int*. Podobná funkcia bola potrebná pre premenlivé hodnoty typu *float*. Premenná typu *float* slúži na skladovanie necelých čísel a potrebuje viac pamäti ako premenná typu *integer*. Funkcia na prevod čísel typu *float* má názov `mapFloat()` a jej základ tvorí zdrojový kód funkcie `map()` [15]. Jediný rozdiel je, že bol zmenený typ vnútorných premenných na *float*. Funkcia patrí do triedy `AutomationShield` a bola použitá v rámci niektorých funkcií v ďalších triedach.

Zdrojový kód 3.4: Funkcia `mapFloat()`

```
float mapFloat(float x, float in_min, float in_max, float
    out_min, float out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min)
        + out_min;
}
```

Prvá funkcia pre Optoshield je kalibračná funkcia s názvom `calibration()` a slúži na zistenie hodnoty fotorezistora pri minimálnej a maximálnej hodnote jasu svetelnej diódy. Je to funkcia bez vstupných parametrov a je typu *void*, čo znamená že nemá žiadne výstupné hodnoty. Zavoláme ju v časti `setup(){ }`, čiže vykoná sa len raz. Kalibračná funkcia trvá minimálne 6-7 sekúnd, potrebujeme čas aby sa ustálila hodnota fotorezistorov a aby odčítané hodnoty boli presné. Presnosť odčítaných hodnôt je nevyhnutná, lebo tieto hodnoty ďalej používame v ďalších funkciách na determinovanie hraníc prevodu. Pracuje s globálnymi premennými pre minimálnu a maximálnu hodnotu, ktoré sú privátne. Ak je sériová komunikácia inicializovaná, tak funkcia na konci vypíše report o zistených hodnotách, čo uvidíme otvorením Serial Monitoru. Funkcia obsahuje premennú s názvom `_indicator`, ktorá je typu *bool* a môže mať hodnotu *true* a *false*. Uplatní sa pri určovaní hraníc pri prevode vo funkcií `sensorRead()`.

Zdrojový kód 3.5: Zdrojový kód funkcie `calibration()`

```
void calibration() {                                // zisti min. a max.
    // hodnotu pre prevod
    Serial.println("Calibration is running...");   // aby sa hodnota snimaca
    delay(3000);                                    // ustalila
    _minVal = analogRead(OPTO_YPIN);               // citanie min. hodnoty
    analogWrite(OPTO_UPIN, 255);                     // citanie max. hodnoty
    delay(3000);
    _maxVal = analogRead(OPTO_YPIN);
    analogWrite(OPTO_UPIN, 0);                      // na konci vypne ledku
    _indicator = true;
```

```

Serial.println("Calibration is done");
} // koniec kalibracie

```

Funkcia `sensorRead()` vráti odčítanú analógovú hodnotu senzora v percentách. Funkcia nemá vstupné parametre. Na prevod používa už popísanú funkciu `mapFloat()`. Ak zavoláme funkciu `calibration()` tak `_indicator` bude *true* a ako hraničné hodnoty pre-vodu budú minimálna a maximálna hodnota zistená pri kalibrácii. V opačnom prípade hraničné hodnoty sú staticky zadefinované.

Zdrojový kód 3.6: Zdrojový kód funkcie `sensorRead()`

```

float sensorRead(){
    _sensorRead = analogRead(OPTO_YPIN);

    if(_indicator){           // ak podmienka plati pouziva
                           // hodnoty z kalibracie
        _sensorValue = AutomationShield.mapFloat(_sensorRead, _minVal
            , _maxVal, 0.00, 100.00);
    }
    else{
        _sensorValue = AutomationShield.mapFloat(_sensorRead, 20.00,
            800.00, 0.00, 100.00);          // v opacnom pripade hodnoty su
                                         // staticky zadefinovane
    }
    return _sensorValue; }

```

Ďalšia dôležitá funkcia je `begin()`. Funkcia `begin()` je typu void a je bez vstupných parametrov. Voláme ju v sekcií `void setup()`, čiže vykoná sa len raz. Úlohou funkcie je nastavenie pinov na Optoshieldu pomocou príkazu `pinMode()`. Pin môže byť typu *OUTPUT – výstup* alebo *INPUT – vstup*. Ako vstupy sa inicializujú analogové piny. V prípade Optoshieldu vstupy sú A0, A1 a A2 a jediný výstup je D3⁴, pin ktorý napája svetelné diódy. Príklad jedného riadku je `pinMode(OPTO_YPIN, INPUT)`.

V hlavičkovom súbore boli zadefinované názvy všetkých použitých pinov pomocou príkazu `#define`, napr. `#define OPTO_YPIN 1` čo je inštrukcia pre komplilátor. Ten počas komplilovania kódu každé slovo `OPTO_YPIN` nahradí číslom 1, čo je číslo analógového pinu kde je zapojený daný pin fotorezistora. Použitím príkazu `#define` na priradenie príslušných názvov a ich následné použitie v kóde robí kód prehľadnejším a čitateľnejším.

Zdrojový kód 3.7: Zdrojový kód funkcie `begin()`

```

void begin(void){           // zadefinovanie pinov
    pinMode(OPTO_YPIN, INPUT);
    pinMode(OPTO_UPIN, OUTPUT);
    pinMode(OPTO_RPIN, INPUT);
    pinMode(OPTO_YAUX, INPUT); }

```

Nasleduje funkcia `sensorReadVoltage()` ktorá vráti analógovú hodnotu fotorezistora vo voltoch. Je to funkcia bez vstupných parametrov. Funkcia používa konštantu *k* na

⁴Ax je x-tý analógový pin kym Dx je x-tý digitálny pin

prevod analógových hodnôt na napätie. Konšanta k je typu *float* a jej hodnota sa rovná $k = 5.00 / 1023.00$. 1023 znamená maximálnu hodnotu čo dokáže vrátiť analógovo-číslicový (AD) prevodník, ktorý je desať bitový, $2^{10} - 1 = 1023$. Päť voltov, čo je maximálne napätie v obvode, vydelíme týmto číslom a ako výsledok, k sa bude rovnať napätiu pripadajúcu na jednu analógovú hodnotu, čo je 0.00488759 V.

Zdrojový kód 3.8: Zdrojový kód funkcie sensorReadVoltage()

```
float sensorReadVoltage() {
    float k = (5.00 / 1023.00); // konstanta pre prevod
    _valueRead = analogRead(OPTO_YPIN);
    _sensorVoltage = _valueRead * k;
    return _sensorVoltage;
}
```

Funkcia `actuatorWrite()` slúži na nastavenie jasu svetelných diód. Je to funkcia typu *void* s jedným vstupným parametrom typu *float* od 0 do 100. Vstupná hodnota prislúcha jasu diód v percentách a generuje zodpovedný PWM signál. Používa funkciu `mapFloat()` na premenu hodnôt z rozsahu 0-100 do rozsahu 0-255. Rozsah PWM signálu je od 0 do 255, lebo je to digitálny signál a atmel ATmega 328 je 8 bitový mikroradič, $2^8 - 1 = 256 - 1 = 255$.

Zdrojový kód 3.9: Zdrojový kód funkcie actuatorWrite()

```
void actuatorWrite(float value) {
    _convertedValue =
        AutomationShield.mapFloat(value, 0.00, 100.00, 0.00, 255.00);

    analogWrite(OPTO_UPIN, _convertedValue);
}
```

Ďalšia funkcia je `sensorAuxRead()` ktorá číta analógovú hodnotu voľného fotorezistora a vráti výsledok vo voltoch. Je to funkcia bez vstupných parametrov a vráti hodnotu napäťia ako typ *float*.

Zdrojový kód 3.10: Zdrojový kód funkcie sensorAuxRead()

```
float sensorAuxRead() {
    float k = (5.00 / 1023.00); // konstanta pre prevod
    _auxRead = analogRead(OPTO_YAUX);
    _auxVoltage = _auxRead * k;
    return _auxVoltage;
}
```

Posledná funkcia triedy OptoClass je `referenceRead()`. Je to funkcia, podobne ako predchádzajúca, bez vstupných parametrov a vráti analógovú hodnotu potenciometra v percentách. Funkciou vrátená hodnota je typu *float* a používame `mapFloat()` na prevod z analógových hodnôt na percentá do rozsahu 0-100.

Zdrojový kód 3.11: Zdrojový kód funkcie referenceRead()

```
float referenceRead(){      // vrati hodnotu potenciometra
    _referenceRead = analogRead(OPTO_RPIN);
    _referenceValue = AutomationShield.mapFloat(_referenceRead
        ,0.00,1023.00,0.00,100.00);
    return _referenceValue;
}
```

Na kontrolu a na uľahčenie ”debuggingu” som napísal funkcie `returnIndicator()`, `returnMinVal()` a `returnMaxVal()`. Obsahujú jeden `return` príkaz, ktorý vráti hodnotu privátnych premenných `_Indicator`, `_minVal`, `_maxVal`. Funkcia `returnIndicator()` je typu `bool`, ostatné dve sú typu `float`. Zdrojový kód jednej z funkcií je zobrazený nižšie:

Zdrojový kód 3.12: Zdrojový kód funkcie referenceRead()

```
bool returnIndicator(){      // vystupom je hodnota premennej
    // _indicator
    return _indicator;
}
```

Pri riešení úloh z oblasti automatizácie presnosť vzorkovania je klúčová. Jazyk Arduino obsahuje štandardné funkcie na časovanie. Tieto funkcie sú `delay()`, `millis()` a `micros()`. Funkcia `delay(čas v ms)` preruší chod programu na nami zadaný čas v milisekundách. Funkcia `millis()` meria uplynulý čas od posledného resetu dosky v milisekundách. Jediná nevýhoda funkcie je, že pretečie po zhruba päťdesiatich dňoch [7]. Podobne funguje aj funkcia `micros()`, ktorá pretečie po sedemdesiatich minútach [8]. Po pretečení premennej, do ktorej funkcia uklada svoje hodnoty, program sa správa nepredvídateľne, preto na naše účely tieto funkcie nevyhovujú.

Presne vieme vzorkovať použitím Timer registrov a prerušení. Arduino UNO má jeden 16 bitový register, Timer1 a dva 8 bitové registre Timer0 a Timer2. Na vzorkovacie účely najviac vyhovuje 16 bitový register Timer1, ktorý je používaný aj knižnicou Servo - pri súčasnom používaní sa môžu vyskytnúť problémy [9], [10]. Na časovanie som napísal funkciu ktorá sa volá `Sampling()`.

Funkcia `Sampling(vzorkovací čas)` používa Timer1, je typu void a má jeden vstupný parameter - vzorkovací čas v milisekundách. Funkcia pracuje s mikrosekundami a preto zadaný vzorkovací čas je vynásobený tisícimi. Maximálna hodnota, ktorú register môže nadobúdať je 65535 ($2^{16}-1$). Túto hodnotu vieme zvýšiť použitím takzvaných preškálovačov - prescaler. Základná hodnota preškálovača je 1^5 . Slúžia na zvýšenie hodnoty registra takým spôsobom, že hodnota registra sa zvýší o jednu po každom x-tom počítaní, kým x je hodnota použitého preškálovača. Pri použití preškálovača 8 hodnota registra sa zvýší o jednu po každom ôsmom počítaní. Použitie preškálovačov umožňuje aplikáciu väčšieho vzorkovacieho času.

Rozlíšenie pri jednotlivých preškálovačoch dostanene vydelením hodnoty preškálovača s frekvenciou mikroradiša, čo je 16 MHz pri Arduino UNO. Výsledok bude v mikrosekundách. Vzorový výpočet obsahuje Rovn. 3.10 na strane 24.

⁵Ďalšie možné hodnoty sú 8, 64, 256 a 1024.

$$R = \frac{64}{16} = 4 \text{ ms.} \quad (3.10)$$

Platí, že volíme si vhodný preškálovač tak, že keď vydelíme vzorkovací čas v mikrosekundách rozlíšením preškálovača, výsledok musí byť menší ako 65535. V prípade funkcie `Sampling()` možné rozlíšenia pre preškálovače sú uložené v poli s názvom *resolution* a jeden for cyklus postupne skúša hodnoty. Ak nájde menšiu hodnotu ako 65535, podľa poradového čísla v poli a s pomocou switch-case si nastaví bity vhodného preškálovača. Informácie o poliach a ich používaní nájdete na stránke [13].

Zdrojový kód 3.13: Výber vhodného preškálovača

```

float Tr1 = 0.0625;           // rozlisenie pri p = 1
float Tr8 = 0.5;              // rozlisenie pri p = 8
float Tr64 = 4;               // rozlisenie pri p = 64
float Tr256 = 16;              // rozlisenie pri p = 256
float Tr1024 = 64;             // rozlisenie pri p = 1024

float resolution[] = {Tr1, Tr8, Tr64, Tr256, Tr1024};
                           // pole obsahujuce preskalovace

float uTs = (float(sampleTime) * 1000.00);
                           // vzorkovaci cas

                           // cyklus vyberie spravny
                           // preskalovac
for(_i = 0; _i < 5; _i++){
    _testValue = uTs / resolution[_i];
    if(_testValue <= Lt){
        break;
    }                               // koniec podmienky
}                               // koniec cyklu

switch (_i){

case 0:
TCCR1B |= (1 << CS10); // preskalovac 1 (zaklad)

TCCR1B |= (0 << CS11); // nastavenie bitov na 0
TCCR1B |= (0 << CS12);
break;

case 1:
TCCR1B |= (1 << CS11); // prescaler 8

TCCR1B |= (0 << CS10);
TCCR1B |= (0 << CS12);
}

```

```

break;

case 2:
TCCR1B |= (1 << CS10); // prescaler 64
TCCR1B |= (1 << CS11);

TCCR1B |= (0 << CS12);
break;

case 3:
TCCR1B |= (1 << CS12); // prescaler 256

TCCR1B |= (0 << CS10);
TCCR1B |= (0 << CS11);
break;

case 4:
TCCR1B |= (1 << CS10); // prescaler 1024
TCCR1B |= (1 << CS12);

TCCR1B |= (0 << CS11);
break;

default:
Serial.println("The sampling time is higher than the
limit, please try smaller value");

}

```

Ostatné nepoužité bity sú nastavené na nulu. Na začiatku všetky bity sú resetované a prerušovania nie sú povolené.

Zdrojový kód 3.14: Resetované bity na začiatku

```

TCCR1A = 0;                                // resetuje bity
TCCR1B = 0;
TCNT1 = 0;
cli();                                     // prerusenia su zakazane

```

Vhodným nastavením bitov si vyberieme CTC mód, čo znamená, že vzorkuje pri resetnutí OCR1A registra, ktorý obsahuje vypočítanú hodnotu `_testValue` podľa zadaného vzorkovacieho času, zníženú o jednu.

Zdrojový kód 3.15: Nastavenie CTC módu

```

OCR1A = _testValue - 1;      // max. hodnota pred resetom

```

```

TCCR1B |= (1 << WGM12);    // CTC mod

```

Po nastavení preškálovača a vhodného vzorkovacieho módu funkcia povolí globálne prerušenia.

Zdrojový kód 3.16: Povolené prerusenia

```
TIMSK1 |= (1 << OCIE1A); // povoli Timer prerusenia  
sei(); // povoli globalne prerusenia
```

Používame premennú **StepEnable** ktorá je typu *bool* a základná hodnota je *false*. Ak sa OCR1A resetne, hodnota **StepEnable** sa mení na *true*.

Zdrojový kód 3.17: StepEnable

```
ISR(TIMER1_COMPA_vect){ // ak sa resetne register  
// StepEnable bude true  
StepEnable = true;  
}
```

Vzorkovať vieme použitím podmienky **if(StepEnable){ x }** ktorú dáme do časti **loop()** a vykoná sa kód medzi zátvorkami v prípade, ak hodnota premennej **StepEnable()** sa mení na **true**.

Zdrojový kód celej funkcie sa nachádza medzi prílohami, ako príloha A.

4 Motoshield

4.1 Hardware

4.1.1 Popis súčiastok

Druhým zo série plošných spojov je Motoshield. Akčným členom je jednosmerný motorček a meranie otáčok je vyriešený pomocou enkódera s Hallovou sondou. Vstup do systému $u(t)$ predstavuje energia dodaná motoru a výstup zo systému $y(t)$ predstavujú otáčky hriadeľa motora.

Používaný jednosmerný motorček je vyrobený firmou DFROBOT. Jeho rozmery sú ($d \times š \times v$) $40.5 \times 14.5 \times 12$ milimetrov a váži 18 gramov. Pracovné napätie motora je 6 voltov a prúdový odber podľa výrobcu je 60 mA v nezaťaženom stave pri šiestich voltoch. Menovitý moment je $2.5 \text{ kg}^*\text{cm}$ a menovité otáčky sú 24 ot/min. Prevodový pomer je 380:1, to znamená, že kým raz sa otočí predný hriadeľ zadný hriadeľ spraví 380 otáčok [2].

Motorová jednotka má šesť vývodov: M1, GND, C1, C2, VCC a M2. Samotný motor má dva vývody, M1 a M2, cez ktorých je napájaný. Ostatné vývody patria k enkóderu. VCC a GND slúžia na napájanie, C1 a C2 predstavujú kanály enkódera. Enkóder sa skladá z dvoch Hallových sond. Hallova sonda je senzor ktorý je aktivovaný externým magnetickým poľom. Výstupný signál Hallovej sondy je funkciou hustoty magnetického toku okolo zariadenia. Ak hodnota hustoty magnetického toku okolo senzora presahuje určitú preddefinovanú hodnotu tak senzor generuje výstupné napätie [3]. Hallova sonda je určená na meranie otáčok a nachádza sa na zadnej strane motorovej jednotky. Koliesko pripojené na zadný hriadeľ obsahuje sedem pólových párov. Točením sa zadného kolieska detektor sníma jednotlivé póly. Pri detekcii severného pólu Hallova sonda generuje signál zodpovedný hodnote logickej jednotky a pri detekcií južného pólu generuje signál zodpovedný hodnote logickej nuly. Zmeny hodnôt signálu môžu byť sledované pomocou prerušení.

Prerušenia sú bežne používané v programátorskej praxi, podľa zadefinovaných podmienok čakajú v pozadí a keď podmienky sú splnené vykoná sa určitý kód. Tieto podmienky sa vzťahujú k zmenám výstupného signálu z Hallovej sondy. V prípade Motoshieldu bol použití spúšťač FALLING ktorý sa aktivuje pri zmene signálu z vysokej úrovni na nízku úroveň. Samozrejme môžu byť použité aj ďalšie spúšťače podľa potreby, ďalšie informácie o prerušení sú na oficiálnej stránke Arduina [4]. Prerušenia sa nevykonávajú cyklicky ako bežný program a ak prerušenie beží tak základný program prestane behať. Ak kód v rámci prerušenia bol vykonaný, základný program pokračuje v behu. Preto nie je odporúčané písanie dlhé kódy do prerušení ani používať príkazy ako Serial.println(). Väčšinou sa

tam nachádza jedna premenná ktorej hodnota sa zvyšuje o jednu pri splnených podmienkach. Arduino UNO má vyhradené dva digitálne piny pre prerušenia a tie sú D2 a D3. Spustenie prerušenia si nastavíme v časti ISR – Interrupt Service Routine nasledovne: `attachInterrupt(digitalPinToInterrupt(číslo_pinu), názov_funkcie, typ_spúšťača);` Do časti `digitalPinToInterrupt()` si napíšeme číslo digitálneho pinu na ktorý je pripojená Hallova sonda. Potom nasleduje názov funkcie ktorá sa má vykonať a typ spúšťača.

Pomocou premennej si sledujeme otáčky predného hriadeľa motora. Vieme, že jedna otáčka predného hriadeľa predstavuje 380 otáčok zadného kolieska. Toto číslo si vynásobíme metódou spúšťača (Pri metódach FALLING alebo RISING je to 1, pri metóde CHANGING 2) a počtom pôlových párov. Výsledok je 2660, čo predstavuje jednu otáčku predného hriadeľa.

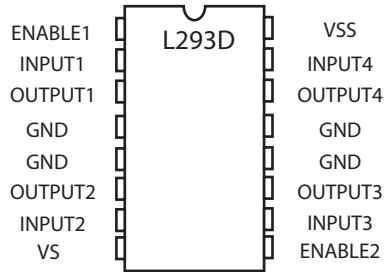
$$Rev = 380 * 1 * 7 = 2660 \text{ hod/ot.} \quad (4.1)$$

Samotný motorček nemôže byť priamo pripojený k Arduinu, lebo odoberá také množstvo prúdu ktorý by poškodil mikropočítač. Motor som pripojil nepriamo pomocou súčiastky, ktorá dostane malý prúdový impulz od mikropočítača a sprostredkuje motoru už tak zosilnený impulz, že veľkosť prúdu stačí na ovládanie motora. Túto úlohu splní aj obyčajný tranzistor ale má jednu nevýhodu - nie je možná regulácia smeru otáčania. Preto na Motoshield bol používaný integrovaný obvod L293D v štandardnom DIP16 prevedení, ktorý sa bežne používa v hračkách a v hobby projektoch. Čip s pomenovanými vývodmi je zobrazený na Obr. 4.1. Do čipu L293D je integrovaný štvorkanálový H-mostík. Každý z kanálov je schopný ovládať cievku alebo relé. Dva kanále sú potrebné na ovládanie jednosmerných motorčekov, smer otáčania je meniteľný. Na ovládanie krokových motorov potrebujeme všetky štyri kanály. Dokáže ovládať aj unipolárne aj bipolárne krokové motory [5],[11].

Charakteristiky čipu [5]:

- Každý z kanálov poskytuje 600 mA prúdu čo vo väčšine prípadov vyhovuje na ovládanie jednosmerných motorčekov. Ďalej, každý kanál dokáže krátkodobo a neopakovateľne sprostredkovať prúd veľkosti až 1.2 A, lebo motor pri spustení má väčší prúdový odber. Ak jednosmerný motorček dlhodobo alebo viac krát za sebou bude ťahať 1.2 A alebo viac, môže poškodiť čip.
- Napájacie napätie môže mať hodnotu až 36 V.
- Ochrana proti prehriatiu, vnútorný senzor sníma teplotu a ak teplota dosiahne určitú prednastavenú hodnotu tak čip prestane ovládať motor.
- Model L293D má vstavané vnútorné diódy ktoré chránia proti napäťovým špičkám pri vypínaní a zapínaní cievok motoru.

Na ovládanie jednosmerného motora stačí jedna strana čipu. Na Motoshield je použitá ľavá strana. Pomocou prvého Enable pinu je nastavená rýchlosť motora. Ak na pinu



Obr. 4.1: Integrovaný obvod L293D s pomemovanými vývodmi

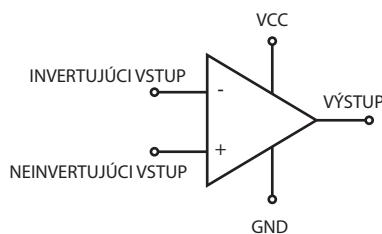
Tabuľka 4.1: Pravdivostná tabuľka integrovaného obvodu L293D

ENABLE1	INPUT1	INPUT2	VÝSTUP
HIGH	HIGH	LOW	Točí sa reverzne - proti smeru chodu hodinových ručičiek
HIGH	LOW	HIGH	Točí sa v smere chodu hodinových ručičiek
HIGH	HIGH	HIGH	Netočí sa
HIGH	LOW	LOW	Netočí sa
LOW	X	X	Netočí sa

Enable nie je napätie , tak výstupy čipu sú vypnuté, bez ohľadu na napäťový stav vstupných pinov a motor sa neotáča. Tento pin bol pripojený na piaty digitálny pin Arduina, ktorý je schopný generovať aj PWM signál. Druhý a siedmy pin sú INPUT1 a INPUT2. Tieto boli pripojené na D6 a D7, slúžia na nastavenie smeru otáčania hriadeľa motora. Pravdivostnú tabuľku kombinácií možete vidieť v Tab. 4.1 na strane 29. Schéma zapojenia je zobrazená na Obr. 4.7 na strane 33.

Tretí a šiesty pin sú OUTPUT1 a OUTPUT2. Pomocou tretieho a šiesteho pinu je motor pripojený k čipu. Štvrtý a piaty pin sú uzemnené. Všetky GND piny sú vnútorné spojené. Šestnásy pin bol pripojený na napájanie a slúži ako zdroj napäťia pre vnútornú logiku kym ôsmy pin bol tiež pripojený na napájanie ale slúži ako zdroj energie pre motor. Na ôsmy pin môžeme pripojiť až 36 V.

Operačný zosilňovač je lineárna súčiastka používaná na úpravu signálu a na vykonanie matematických operácií. Má dva vstupy, invertujúci (-) a neinvertujúci (+) vstup a má jeden výstup. Základná konfigurácia operačného zosilňovača je rozdielová, odčíta menšiu hodnotu privedeného napäťia z väčšej hodnoty. Má obrovské vnútorné zosilnenie. Na Obr. 4.2 je zobrazená všeobecná schematická značka operačného zosilňovača.



Obr. 4.2: Schematická značka operačného zosilňovača

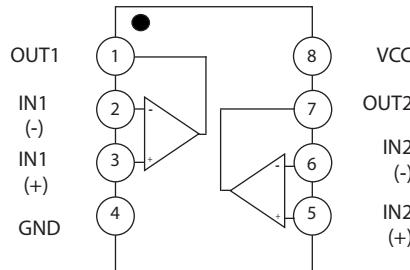
Na reguláciu obrovského zosilnenia používame zápornú spätnú väzbu. Pod pojmom záporná spätná väzba rozumieme rezistor zapojený do invertujúceho vstupu (-) zosil-

ňovača. Zmenou hodnôt rezistora si vieme modifikovať veľkosť zosilnenia.

Dve všeobecné pravidlá platia pre operačné zosilňovače:

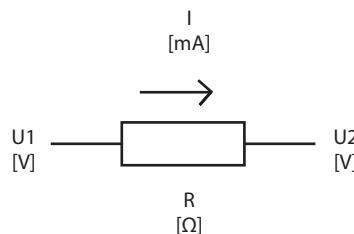
- Žiadny prúd netečie cez vstupy zosilňovača
- Operačný zosilňovač sa snaží vyrovnať hodnoty napäcia na vstupoch

Na Motoshieldie bol použitý integrovaný obvod LM358 ktorý obsahuje dva interné nezávislé operačné zosilňovače. Jeho vnútorná štruktúra je zobrazená na Obr. 4.3.



Obr. 4.3: Operačný zosilňovač LM358

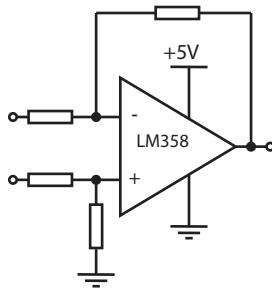
V zapojení sa nachádzajú dva zosilňovače ktoré sa uplatnili pri spracovaní signálu pri meraní prúdového odberu motora. Prúd je meraný pomocou odporu aplikovaním Ohmovho zákona. Do prvého vstupu riadiaceho integrovaného obvodu (OUTPUT1) je zapojený jeden vývod motora, ktorý je prerušený odporom. Na meranie prúdu bol použitý odpor s hodnotou 10Ω a pomocou programu bol sledovaný úbytok napäcia. Podľa Ohmovho zákona úbytok napäcia vydelíme odporom a dostaneme hodnotu tečúceho prúdu cez odpor čo sa zároveň rovná aj prúdovému odberu motora. Pri výpočte používame základné jednotky. Rovnica 4.2 obsahuje vzorový výpočet veľkosti odobratého prúdu jednosmerným motorom podľa Obr. 4.4.



Obr. 4.4: Meranie prúdu

$$I = \frac{U_1 - U_2}{R} = \frac{4.1 \text{ V} - 3.8 \text{ V}}{10 \Omega} = \frac{0.3 \text{ V}}{10 \Omega} = 0.03 \text{ A.} \quad (4.2)$$

Na dvoch vývodoch rezistora sú rôzne hodnoty napäcia U_1 a U_2 . Prvý zosilňovač je zapojený ako rozdielový, odčíta menšie napätie U_2 z väčšej hodnoty napäcia U_1 v prípade, ak platí, že $U_1 > U_2$. Na výstupe bude rodieť dvoch napäti. Väčšie napätie je privedené



Obr. 4.5: Zapojenie operačného zosilňovača na odčítanie hodnôt napäťia

Tabuľka 4.2: Testovanie zapojenia na odčítanie hodnôt 1

Stále napätie = 5.098V

VSTUP (Laboratórny zdroj)	VÝSTUP (OPAMP)
4.0	1.0825
4.1	0.9800
4.2	0.8828
4.3	0.7805
4.4	0.6834
4.5	0.6576
4.6	0.6587

na neinvertujúci vstup (+) zosilňovača, kým to menšie napätie je privedené na invertujúci vstup (-). Schému rozdielového zosilňovača môžete vidieť na Obr. 4.5.

V konfigurácií sa používajú štyri odpory s rovnakou hodnotou a preto nedochádza k žiadnemu zosilneniu výstupného napäťia. V prvej konfigurácii boli použité odpory s hodnotou $10\text{ k}\Omega$. Pri testovaní sa ale ukázalo, že zapojenie nesplní naše očakávania a správne funguje len v určitom rozsahu napäťia. Obvod bol testovaný pomocou externého laboratórneho zdroja s dvomi kanálmi. Z jedného kanála bolo vyvedené napätie $U_1 = 5.098\text{ V}$ a táto hodnota bola stála. Z druhého kanála bolo vyvedené menšie napätie U_2 , ktorej hodnota bola zmenená počas merania. Hodnoty napäti boli skontrolované multimetrom. Počiatočná hodnota U_2 boli 4 V. Hodnota napäťia na výstupe zosilňovača bola 1.0825 V a ďalšie hodnoty U_2 boli zväčšené o 100 mV. Výsledky sú zverejnené v nasledujúcej Tab. 4.2 na strane 31.

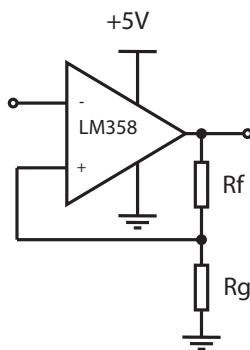
Z tabuľky vyplýva, že pri hodnotách $U_2 = 4.5\text{ V}$ a $U_2 = 4.6\text{ V}$ sme na výstupe dostali skoro rovnaké hodnoty napäťia. Ukázalo sa, že aplikácia väčších odporov čiastočne vyrieši náš problém. Hodnoty odporov boli zmenené na $1\text{ M}\Omega$. Po výmene nasledovalo testovanie obvodu. Výsledky testu vidíte v Tab. 4.3 na strane 32. Z výsledkov vyplýva, že obvod dokáže odčítať hodnoty až do $U_2=5\text{ V}$. Ukázali sa ďalšie obmedzenia obvodu ktorý dokonale funguje až do hodnoty napäťia $U_2=1.3\text{ V}$.

Druhý operačný zosilňovač bol zapojený ako neinvertujúci. To znamená že priebeh vstupného signálu sa nemení, len zvýši sa amplitúda o faktor zosilnenia. Zapojenie je na Obr. 4.6.

Napätie z výstupu prvého (rozdielového) zosilňovača priviedieme na neinvertujúci vstup (+) druhého (neinvertujúceho) zosilňovača. Neinvertujúci zosilňovač je zobrazený na Obr. 4.6.

Tabuľka 4.3: Testovanie zapojenie na odčítanie hodnôt
Stále napätie = 5.098V

VSTUP (Laboratórny zdroj)	VÝSTUP (Laboratórny zdroj)
4.0	1.1017
4.1	0.9990
4.2	0.9012
4.3	0.7984
4.4	0.7004
4.5	0.5975
4.6	0.4947
4.7	0.3967
4.8	0.2940
4.9	0.1960
5.0	0.0935



Obr. 4.6: Zapojenie neinvertujúceho operačného zosilňovača

Medzi invertujúcim vstupom (-) a výstupom druhého zosilňovača je dvojica odporov R_f a R_g , pomocou ktorých si vieme nastaviť faktor zosilnenia nasledovne, viď. Rov. 4.3 na strane 32.

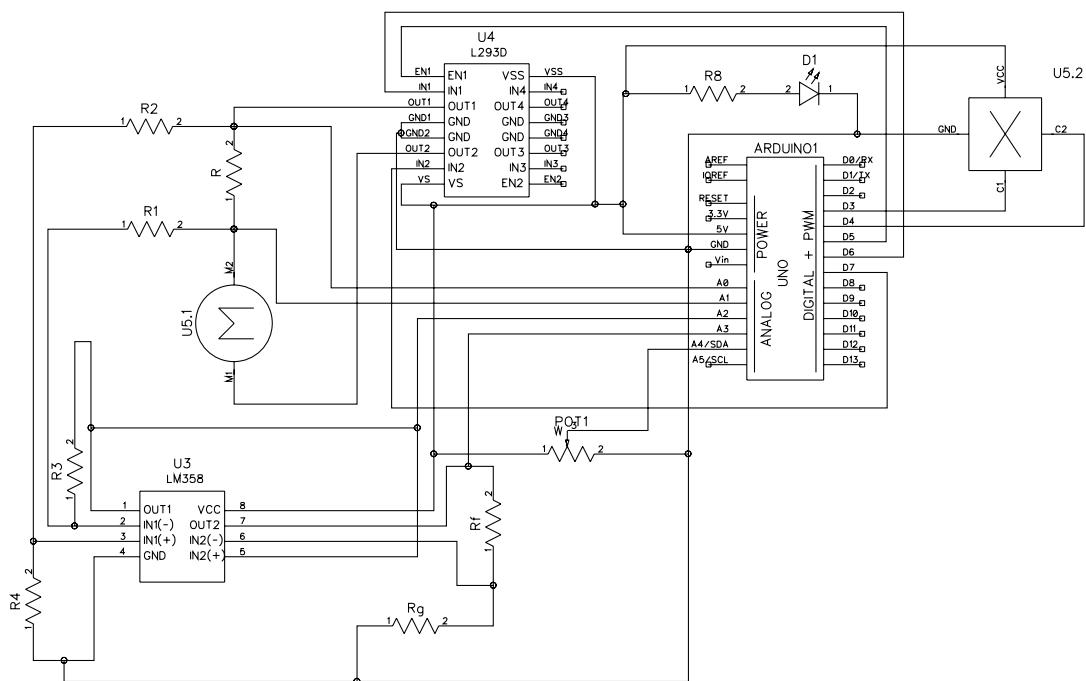
$$A = 1 + \frac{R_f}{R_g} = 1 + \frac{10 \text{ k}\Omega}{5.1 \text{ k}\Omega} = 2.96. \quad (4.3)$$

Na výstupe neinvertujúceho zosilňovača bude už zosilnené napätie ktoré je privedené na analógový pin A2.

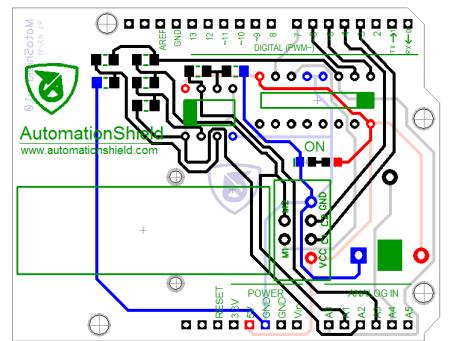
Na uloženie ovládača bola použitá päтика. Päтика môže byť užitočná pri výmene súčiastky, napríklad pri poškodení. Výhoda pätpice je, že poškodenú súčiastku nemusíme odspájkovať, jednoducho si vyberieme a nasunieme novú. Motorček je upevnený továrenskou plastovou objímkou a pomocou dvoch skrutiek s rozmermi M2. Výhoda objímky je, že je sériovo vyrobenná a je dostupná s motorčekom. Pre skrutky boli urobené montážne diery počas navrhovania plošného spoja.

4.1.2 Kreslenie schémy zapojenia a vytvorenie plošného spoja

Schéma bola nakreslená v programe DipTrace. V schéme sa nachádzajú aj mnou vytvorené súčiastky už predstavenou metódou, ktoré teraz tvoria časť knižnice ControlShield. Podľa schémy bol vygenerovaný a zhotovený plošný spoj. Použité rezistory a svetelné diódy sú typu SMD a ich veľkosť je 0805¹. Táto veľkosť je ešte spájkovateľné vlastnoručne bez potreby špeciálneho náradia. Technika spájkovania SMD súčiastok je iná ako technika použitá pri bežných súčiastkách. Kontaktné miestá si ošetríme kolofóniou alebo fixkou, ktorej náplň je podobná kolofónii. Ideálne je používať cín s priemerom 0.5 mm a pomerom 60/40². Na manipuláciu so súčiastkami používame (antistatickú) pinzetu. Treba si dávať pozor pri spájkovaní súčiastok s polaritou.



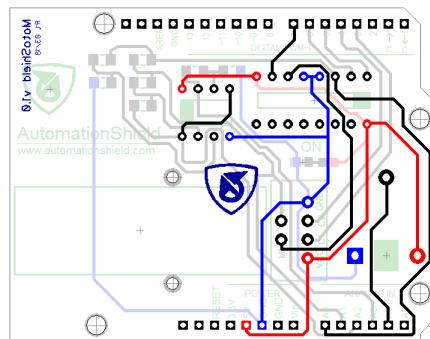
Obr. 4.7: Hotová schéma MotoShield



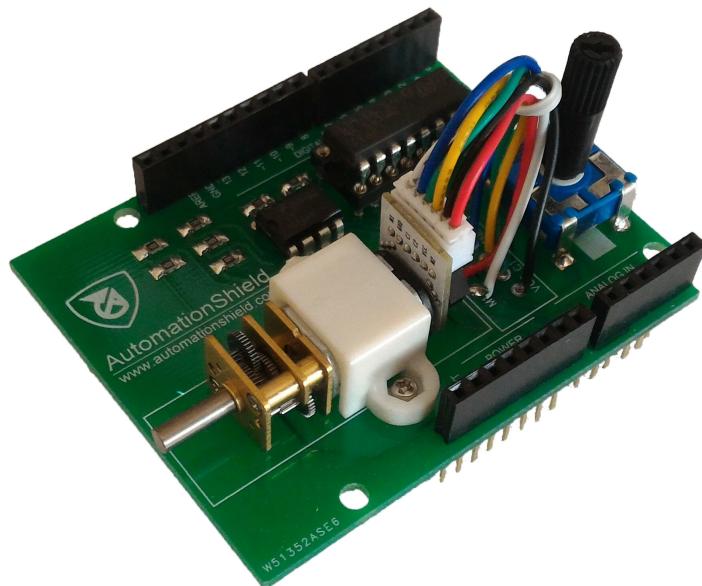
Obr. 4.8: Plošný spoj MotoShield - predná strana

¹2 x 1.2 x 0.45 mm (d x š x v)

²60% Sn a 40% Pb



Obr. 4.9: Plošný spoj MotoShield - zadná strana



Obr. 4.10: Hotový Motoshield

4.2 Software

Táto sekcia sa zaobráva s popisom funkcií napísaných pre Motoshield. Funkcie budú postupne charakterizované a bude popísaný ich pracovný princíp.

Aj táto knižnica sa skladá z hlavičkového (.h) súboru a zo súboru typu .cpp . Hlavičkový súbor sa začína všeobecnou začiatocnou podmienkou ktorá sa zabráni nechcenému viacnásobnému zadefinovaniu knižnice [6]. Ďalej sa tam nachádza príkaz #include "AutomationShield.h" ktorý vloží zdrojový kód knižnice AutomationShield do hlavičkového súboru. Funkcie zadefinované v knižnici AutomationShield sa stanú použiteľným aj v knižnici MotoShield.

Zdrojový kód 4.1: Začiatočná podmienka

```
#ifndef MotoShield_h      // zabrani viacnasobne zadefinovanie
#define MotoShield_h       // zadefinuj ak nie je zadefinovany

#include "AutomationShield.h"

// kod piseme sem

#endif
```

Prvá funkcia je funkcia `begin()`. Je to funkcia typu void bez vstupných parametrov. Slúži na nastavenie pinov použitím hardvérom. Piny nastavuje ako vstupy alebo výstupy pomocou príkazu `pinMode()`. Ďalej rieší nastavenie ISR - Interrupt Service Routine. ISR slúži na nastavenie parametrov prerušení. Ako posledné si nastaví hodnotu statickej globálnej premennej `counter` na nulu. Funkciu voláme v časti `void setup()`.

Zdrojový kód 4.2: Zdrojový kód funkcie `begin()`

```
void begin(void){           // nastavenie pinov
pinMode(MOTO_UPIN,OUTPUT);
pinMode(MOTO_IN1,OUTPUT);
pinMode(MOTO_IN2,OUTPUT);
pinMode(MOTO_C2,INPUT_PULLUP);
pinMode(MOTO_C1,INPUT_PULLUP);
pinMode(MOTO_RPIN,INPUT);
pinMode(MOTO_Vout2,INPUT);
pinMode(MOTO_Vout1,INPUT);
pinMode(MOTO_U1,INPUT);
pinMode(MOTO_U2,INPUT);

attachInterrupt(digitalPinToInterrupt(MOTO_C1),
MotoShield.countTicks, FALLING); // nastavenie ISR

counter = 0;                // nastavenie pociatocnej hodnoty
                           // premennej
} // koniec funkcie begin()
```

Ďalšia funkcia je `countTicks()`. Je to funkcia použitá v ISR. Vykoná sa, ak hranatý signál z enkódera sa zmení z vysokej hodnoty na nízku hodnotu. Vtedy sa hodnota premennej `counter` sa zvýší o jednu. Každá siedma hodnota predstavuje jedno otočenie zadného kolieska. Ďalej sa tam nachádza premenná `Bstate` ktorá je použitá pri určovaní smeru otáčania hriadeľa.

Zdrojový kód 4.3: Zdrojový kód funkcie `countTicks()`

```
void countTicks(){          // ISR - Interrupt Service Routine
Bstate = digitalRead(MOTO_C2);
counter ++ ;
}
```

Nasleduje funkcia `motorON()`. Nastaví rýchlosť motora na maximálnu. Zdrojový kód obsahuje len jeden príkaz typu `analogWrite()`.

Zdrojový kód 4.4: Zdrojový kód funkcie `motorON()`

```
void motorON(){
analogWrite(MOTO_UPIN, 255);
}
```

Samotná funkcia `motorON()` nestačí na zapnutie motora. Po nastaveniu rýchlosť si potrebujeme nastaviť aj smer otáčania. Na to slúži funkcia `setDirection()`. Je to funkcia typu void, čiže bez výstupu ale má jeden vstupný parameter typu *bool*. Ak nami zadaná hodnota je pravdivá - *true*, tak smer otáčania je proti smeru chodu hodinových ručičiek. V opačnom prípade pri zadaní nepravdivej hodnoty *false* hriadeľ sa točí v smere chodu hodinových ručičiek.

Zdrojový kód 4.5: Zdrojový kód funkcie `setDirection()`

```
void setDirection(bool dir){
if(dir){                                // ked dir = true, toci sa proti
    smeru
        // hod. ruciciek
digitalWrite(MOTO_IN1, HIGH);
digitalWrite(MOTO_IN2, LOW);
}
else {                                 // ked dir = false, smer hod.
    // ruciciek
digitalWrite(MOTO_IN1, LOW);
digitalWrite(MOTO_IN2, HIGH);
}
} // koniec funkcie setDirection()
```

Funkcia `revDirection()` otočí už vopred nastavený smer otáčania. Funkcia je typu void a je bez vstupných parametrov.

Zdrojový kód 4.6: Zdrojový kód funkcie `revDirection()`

```
void revDirection(){
Direction = digitalRead(MOTO_IN1);
if(Direction){                         // ak sa toci v smere hod. ruciciek
digitalWrite(MOTO_IN1, LOW);
digitalWrite(MOTO_IN2, HIGH);
}

else{                                  // opacne
digitalWrite(MOTO_IN1, HIGH);
digitalWrite(MOTO_IN2, LOW);
}
} // koniec funkcie revDirection()
```

Funkcia `motorOFF()` vypne motor. Dosiahne to vypnutím *Enable1* pinu ovládača. Ak napäťová hodnota tohto pinu je nízka, tak výstupy ovládača nie sú aktívne.

Zdrojový kód 4.7: Zdrojový kód funkcie motorOFF()

```
void motorOFF(){
digitalWrite(MOTO_UPIN, LOW);
}
```

Napísal som aj funkciu na nastavenie rýchlosi otáčania. Táto funkcia sa volá `setMotorSpeed()`. Vstupná hodnota je rýchlosť motora v percentách, v rozsahu 0-100. Táto hodnota je konvertovaná funkciou `mapFloat()` z knižnice AutomationShield do rozsahu 0-255. Výsledná hodnota je napísaná na správny pin (EN1) príkazom `analogWrite()`.

Zdrojový kód 4.8: Zdrojový kód funkcie setMotorSpeed()

```
void setMotorSpeed(float value){
if(value < 30){ value = 30; } // minimum aby sa hriadele tocili
convertedValue = AutomationShield.mapFloat(value
, 0.00, 100.00, 0.00, 255.00);
analogWrite(MOTO_UPIN, convertedValue);
}
```

Funkcia `referenceRead()` číta hodnoty potenciometra a načítané hodnoty vráti v percentách. Na prevod používa funkciu `mapFloat()` z knižnice AutomationShield. Zdrojový kód je rovnaký ako pri OptoShielde.

Funkcia `readVoltage()` určí pokles napäťia na rezistore R. Nemá vstupné parametre a vráti hodnotu určeného napäťia. Pracuje na princípe snímania analógových hodnôt z dvoch vývodov rezistora a následne určí ich rozdiel. Rozdiel je vynásobený konštantou, ktorej hodnota je $k = 5 / 1023$. Po vynásobení výsledok bude vo voltoch.

Zdrojový kód 4.9: Zdrojový kód funkcie readVoltage()

```
float readVoltage(){
ADC1 = analogRead(MOTO_U1);
ADC2 = analogRead(MOTO_U2);

if(ADC1 > ADC2){
ADCU = ADC1 - ADC2;
}

if(ADC2 > ADC1){
ADCU = ADC2 - ADC1;
}

k = (5.00 / 1023.00); //konstanta na prevod

V = ADCU * k;

return V;
} // koniec funkcie readVoltage()
```

Funkcia `readCurrent()` vráti prúdový odber motora v miliampéroch. Vo vnútri je inicializovaná funkcia `readVoltage()` ktorá určí hodnotu napäťia. Potom podľa Ohmovho

zákona veľkosť prúdu sa rovná rozdielu poklesu napäťa a hodnoty odporu rezistora. Čiastočný výsledok je vynásobený tisícmi aby konečný výsledok je v miliampéroch.

Zdrojový kód 4.10: Zdrojový kód funkcie readCurrent()

```
float readCurrent(){      // vrati hodnotu prudu v mA
R = 10;
float callVolt = MotoShield.readVoltage();
I = (callVolt / float(R)) * 1000.00;
                           // *1000.00 aby hodnoty boli v mA

return I;
} // koniec funkcie readCurrent()
```

Funkcia durationTime() vráti čas jedného otáčania sa hlavného hriadeľa v milisekundách. Čas je sledovaný funkciou millis(). Funkcia sleduje hodnotu globálnej premennej counter. Vieme, že po jednej otáčke zadného kolieska hodnota sa zvýši o 380. Ďalšia premenná s názvom previousCount skladuje predchádzajúcu hodnotu počítadla. Z aktuálnej hodnoty je odčítaná predchádzajúca a ak výsledok je väčší alebo rovný 380, zaznamenaná aktuálny čas. Predchádzajúca hodnota času je tiež skladovaná pomocou premennej prevTime(). Na konci z hodnoty aktuálneho času je odčítaná predchádzajúca hodnota a výsledok je vrátený funkciou. Vrátený výsledok predstavuje čas jednej otáčky hriadeľa.

Zdrojový kód 4.11: Zdrojový kód funkcie durationTime()

```
float durationTime(){      // vrati otacky v ms
rev = 380;                  // jedna otacka
count = counter;
cValue = count - previousCount;

t = millis();                // meranie casu

if(cValue >= rev){          // porovnanie hodnot
revTime = t;
durTime = revTime - prevTime;
previousCount = count;
}
prevTime = revTime;

return durTime * 7;
} // koniec funkcie durationTime()
```

Nasledujúca funkcia je readRevolutions(), vráti otáčky za minútu. Sleduje hodnotu premennej counter v istom vzorkovacom čase. Vzorkovací čas je nastavený funkciou Sampling() a tvorí vstupný parameter funkcie int Time. Príkazom noInterrupst() zakážeme prerušenia počas behu funkcie. Pomocou operácie rValue = Count - prevC; zistíme o koľko sa zvýšila hodnota premennej counter za daný vzorkovací čas. Funkcia pracuje na nasledovnom princípe: sleduje koľko otáčok spraví motor v jednej període a to roznásobí na celú minútu. Konšanta h predstavuje číslo, koľkokrát prebehne funkcia

za jednu sekundu. Toto číslo treba vynásobiť šestdesiatimi aby sme dostali koľkokrát prebehne funkcia za minútu a celé vydelíme hodnotou 2660, čo predstavuje jednu otáčku predného hriadeľa motora. Táto hodnota je konštantná a touto konštantou si vynásobíme `rValue` a ako výsledok dostaneme otáčky motora za minútu. Na konci príkazom `interrupts()` povolíme prerušenia.

Zdrojový kód 4.12: Zdrojový kód funkcie `readRevolutions()`

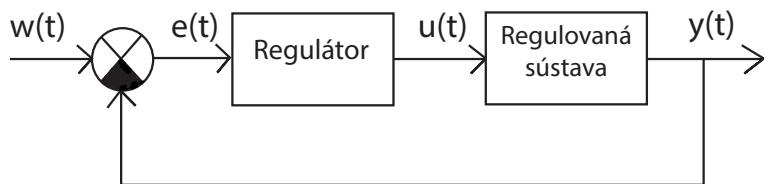
```
float readRevolutions(int Time){  
    Count = counter;  
    noInterrupts();  
    rValue = Count - prevC;  
    h = 1000 / Time;           // constant  
  
    constant = ((float(h) * 60.00) / 2660.00);  
  
    REV = float(rValue) * constant;  
    prevC = Count;  
    interrupts();  
  
    return REV;  
} // end of the readRevolutions() function
```

5 Didaktické príklady

K jednotlivým shieldom patria pripravené príklady, ktoré znázorňujú ich funkčnosť a zároveň vyzdvihujú jednu metódu z oblasti riadenia s ktorou sa študenti môžu zoznámiť. Nasledujúca kapitola sa zaoberá s aplikáciou didaktických príkladov. Obsahuje popis pripravených príkladov, vysvetluje ciele príkladu a obsahuje zdrojový kód každého príkladu. Pre každý shield bude napísaný príklad ktorý sa zaoberá s reguláciou pomocou PID regulátora. Aby študent mohol benefitovať z príkladu musí mať základné vedomosti z oblasti riadenia a o regulátoroch.

5.1 Regulácia a PID regulátor

Regulácia je riadenie so spätnou väzbou. Regulačný obvod sa skladá z riadiaceho systému (regulátor) a z riadeného systému (regulovaná sústava). Regulátor reguluje regulovanú sústavu podľa regulačnej odchýlky $e(t)$, t.j. vstupom do regulátora je regulačná odchýlka. Regulačná odchýlka sa rovná rozdielu žiadanej hodnoty $w(t)$ a regulovanej veľičiny $y(t)$. Regulovaná veličina $y(t)$ je výstupom zo snímača. Podľa vstupných údajov regulátor generuje akčný zásah $u(t)$ ktorým pôsobí na regulovanú sústavu [17].



Obr. 5.1: Regulačný obvod

PID regulátor je v praxi často používaný priemyselný regulátor. Cieľom PID regulátora je udržiavanie výstupu $y(t)$ na užívateľom zadanej úrovni riadenými akčnými zásahmi. Regulátor sa skladá z troch hlavných zložiek: proporcionálnej zložky P, integračnej zložky I a derivačnej zložky D.

Proporcionálna zložka - Akčný zásah proporcionálnej zložky je úmerný regulačnej odchýlke, ktorá je vynásobená zosilnením proporcionálnej zložky K_P . Ak regulačná odchýlka je veľká a kladná, proporcionálna zložka generuje veľký a záporný akčný zásah. Pri nulovej regulačnej odchýlke generuje nulový akčný zásah a pri zápornej regulačnej odchýlke generuje kladný akčný zásah. Je zodpovedný za aktuálnu hodnotu regulačnej odchýlky.

Integračná zložka - Používa sa na odstránenie trvalej regulačnej odchýlky. Čím menšia je hodnota integračnej zložky tým rýchlejšie reaguje na zmeny, ale zároveň stúpa riziko rozkmitania sa sústavy. Berie ohľad na minulé hodnoty regulačnej odchýlky - ak veľkosť akčného zásahu nie je postačujúca na odstránenie regulačnej odchýlky, tak pôsobí na regulátor aby zväčšíl veľkosť akčného zásahu [18].

Derivačná zložka - Veľkosť akčného zásahu generovanej derivačnou zložkou závisí od zmeny regulačnej odchýlky. Čím väčšia je hodnota zmeny tým väčší bude akčný zásah.

Zadefinované zložky môžeme kombinovať. V praxi sa najčastejšie používa P, PI, PD a PID regulátor.

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt} = K_P [e(t) + \frac{1}{T_I} \int_0^t e(\tau) d\tau] + T_D \frac{de(t)}{dt}. \quad (5.1)$$

Na Rov. 5.1 je znázornená rovnica PID regulátora, kde

- K_P - proporcionálna konštanta regulátora
- K_I - integračná konštanta
- K_D - derivačná konštanta
- T_I - integračná časová konštanta
- T_D - derivačná časová konštanta

5.2 Skoková funkcia pomocou Optoshieldu

Odozvu systému môžeme skúmať z časového a z frekvenčného hľadiska. Ak výstup systému sa s časom mení, hovoríme o časovej odozve. Časová odozva sa skladá z dvoch častí: z prechodovej a zo stálej časti. Rozlišujeme tri druhy časových funkcií¹ na testovanie dynamických vlastností systému podľa odozvy [19], [20].

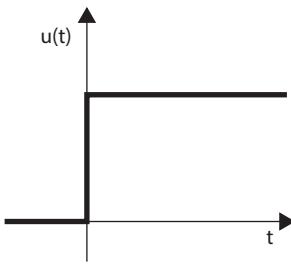
$$u(t) = u_{PR}(t) + u_{ST}(t). \quad (5.2)$$

- $u_{PR}(t)$ - prechodová časť
- $u_{ST}(t)$ - stála časť

Tento príklad sa zaoberá so skokovou funkciou, ktorá predstavuje náhly impulz konštantnej veľkosti na vstup systému (jednotkový skok). Systém na impulz reaguje a nás zaujíma krievka výstupných hodnôt, ktoré postupne dosiahnu nastavenú hodnotu impulzu. Priebeh krievky odozvy závisí od vlastnosti systému. Pomocou skokovej odozvy vieme určiť parametre PID regulátora.

Na Optoshielde skokový experiment je realizovaný nastavením jasu svetelných diód na nami zadanú úroveň a následným sledovaním hodnôt fotorezistora. Nastavenú hodnotu jasu som nazval ako "Setpoint" a predstavuje jas svetelných diód v percentách. Hodnota

¹Impulzná, Skoková a Rampova



Obr. 5.2: Skoková funkcia

je v rozsahu 0-100. Nami nastavenú hodnotu jasu aplikujeme funkciou `OptoShield.actuatorWrite(Setpoint)` v časti `void setup()`. Pomocou funkcie `OptoShield.sensorRead()` sledujeme hodnoty fotorezistora. Získané hodnoty môžeme zbierať do .txt súboru pomocou programu², vo formáte CSV³ a následne zobraziť v MATLABE alebo hodnoty môžu byť priamo zobrazené v programe Arduino IDE, "Serial Plotter"-om. Časovanie je vyriešené funkciou `Sampling()`.

Funkčná časť zdrojového kódu obsiahnutá vo funkcií `step()` sa nachádza nižšie, kým zdrojový kód celého príkladu môžete nájsť medzi prílohami, ako prílohu B.

Zdrojový kód 5.1: Zdrojový kód skokovej funkcie na Optoshield

```
void step(){
float Senzor = OptoShield.sensorRead();

Serial.print(Setpoint);
Serial.print(",");
Serial.println(Senzor);

}
```

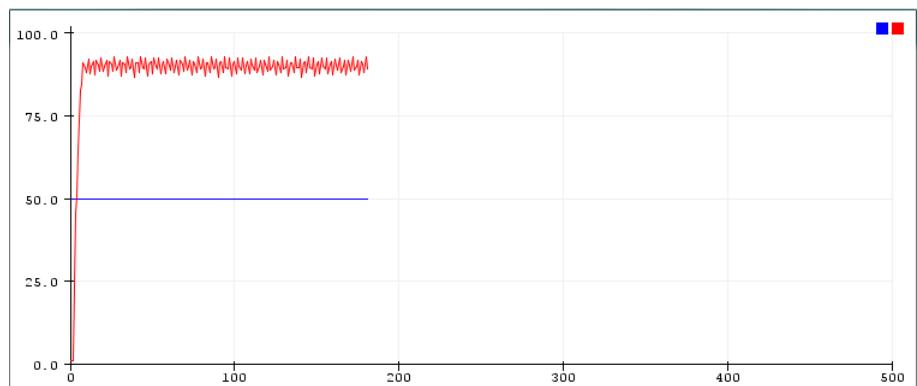
Výsledky testov z pohľadu užívateľa (cez Arduino IDE) sú znázornené na Obr. 5.3 a na Obr. 5.4 na strane 43, kde modrou farbou je označená nami nastavená hodnota jasu a červenou farbou je označený výstup zo senzora. Na Obr. 5.3 môžeme vidieť typické správanie sa systému, pri 50 percentného jasu svetelných diód dostaneme zo senzora skoro maximálnu hodnotu.

Na Obr. 5.5 na strane 43 môžete vidieť odozvu Optoshieldu na skokovú funkciu, zobrazenú v programe MATLAB⁴. Graf zobrazí výstupné hodnoty zo senzora po dobu 2s. Z grafu vyplýva, že prechodová časť krivky je do doby 1s, čiže sekundu trvá senzoru, kým dosiahne nami nastavenú maximálnu hodnotu.

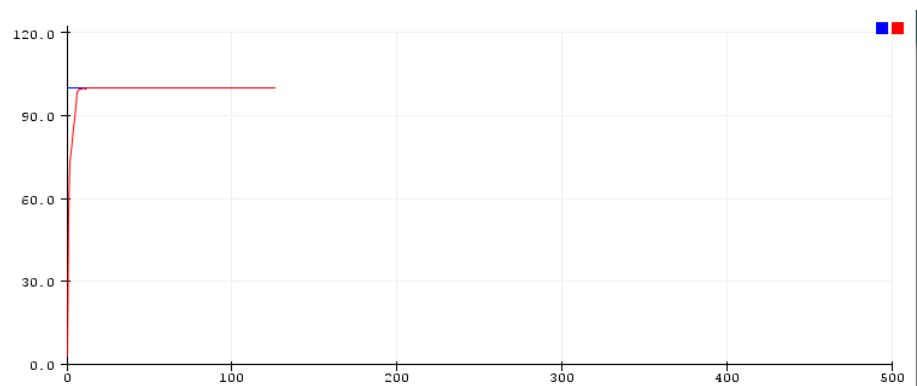
²napr. CoolTerm

³Comma Separated Values

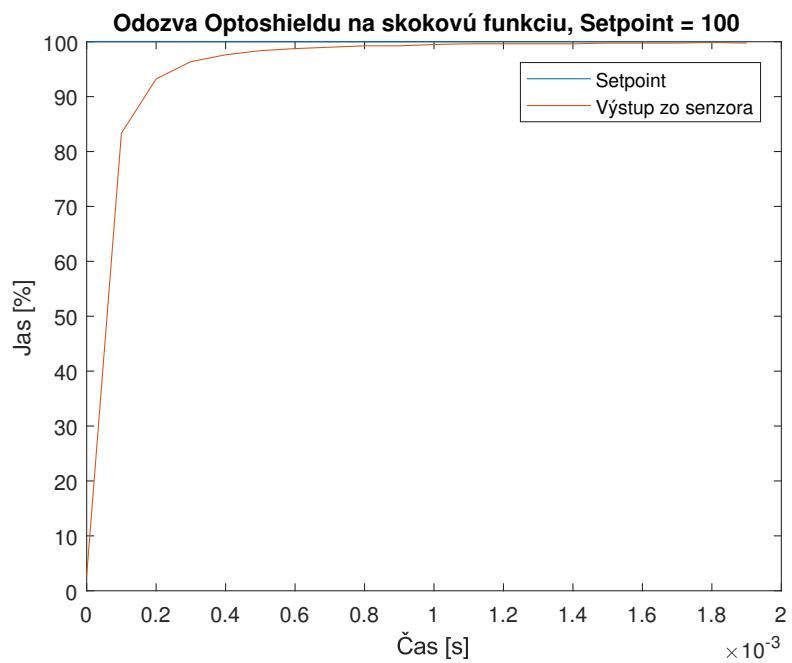
⁴MATrix LABoratory



Obr. 5.3: Odozva Optoshieldu na skokovú funkciu, Setpoint = 50



Obr. 5.4: Odozva Optoshieldu na skokovú funkciu, Setpoint = 100



Obr. 5.5: Odozva Optoshieldu na skokovú funkciu, Setpoint = 100

5.3 PID regulácia pomocou Optoshieldu

Základné informácie o PID regulácii nájdete v Kap. 4.1. Tento príklad aplikuje PID reguláciu na Optoshielde. Cieľom príkladu je nastavenie hodnoty fotorezistora (snímača) na nami zadanú referenčnú úroveň riadenými akčnými zásahmi. Referenčnú úroveň nastavíme pomocou potenciometra. Vhodným nastavením konštant regulátora dbáme na to, aby snímač reagoval čo najrýchlejšie a aby výstupný signál bol čo najhladší a bez prekmitov. Na časovanie používame funkciu `Sampling()`. Zdrojový kód príkladu sa nachádza medzi prílohami pod názvom B.

V časti `void setup()` začíname inicializáciu sériovej komunikácie a dosky Optoshield. Pokračujeme kalibráciou dosky funkciou `OptoShield.calibrate()`. Vzorkovací čas nastavíme pomocou funkcie `Sampling.interruptInitialize(Ts * 1000)`; kde `Ts` je vzorkovací čas v milisekundách a je typu *unsigned long*. Ďalšou funkciou `Sampling.setInterruptCallback(stepEnable)`; si nastavíme funkciu na "Interrupt Service Routine" - ISR. Nižšie sú nastavené parametre regulátora, ktoré sme určili identifikáciou systému alebo inou metódou (napríklad pomocou skokovej odozvy).

Zdrojový kód 5.2: Zdrojový kód príkladu PID - časť `setup()`

```
void setup() {  
  
    Serial.begin(9600);  
  
    OptoShield.begin();           // inicializacia dosky  
    OptoShield.calibration();   // kalibracia  
  
    Sampling.interruptInitialize(Ts * 1000);    // nastavenie  
                                                // vzorkovacieho  
                                                // casu  
    Sampling.setInterruptCallback(stepEnable); // nastavenie ISR  
  
    // nastavenie parametrov PID  
    PIDAbs.setKp(0.1);  
    PIDAbs.setTi(0.015);  
    PIDAbs.setTd(0.0000008);  
  
} // end of the setup
```

Ak vzorkovací čas uplynie a premenná *StepEnable* bude mať hodnotu *true* spustí sa funkcia `step()`, preto sa v nej nachádza funkčná časť programu. Vypočíta regulačnú odchýlku $e(t)$, ktorá je rozdielom referencie z potenciometra a výstupu zo senzora. Je uložená do premennej *error*. Veľkosť akčného zásahu vypočíta pomocou funkcie `PIDAbs.compute(error, 0, 100, 0, 100)`. Funkcia má 5 vstupných parametrov. Prvým parametrom je premenná regulačnej odchýlky. Nasleduje dvojica, ktorá determinuje oblasť, v ktorej sa má zobraziť výstup a posledná dvojica determinuje rozsah pre "anti-windup", čo je ochrana proti nasýteniu integračnej zložky. Vypočítaný akčný zásah aplikujeme funkciou `OptoShield.actuatorWrite(u)`.

Zdrojový kód 5.3: Zdrojový kód príkladu PID - časť step()

```
void step(){

    r = OptoShield.referenceRead();      // citanie referencnej
                                         // hodnoty potentiometra
    y = OptoShield.sensorRead();        // citanie hodnoty senzora

    error = r - y;                   // regulacna odchylka

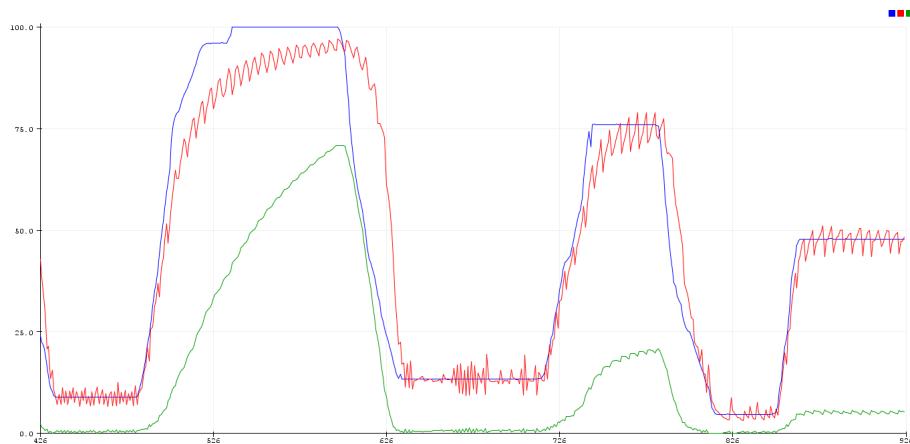
    u = PIDAbs.compute(error,0,100,0,100);
                                         // pocitanie akcneho zasahu

    OptoShield.actuatorWrite(u);

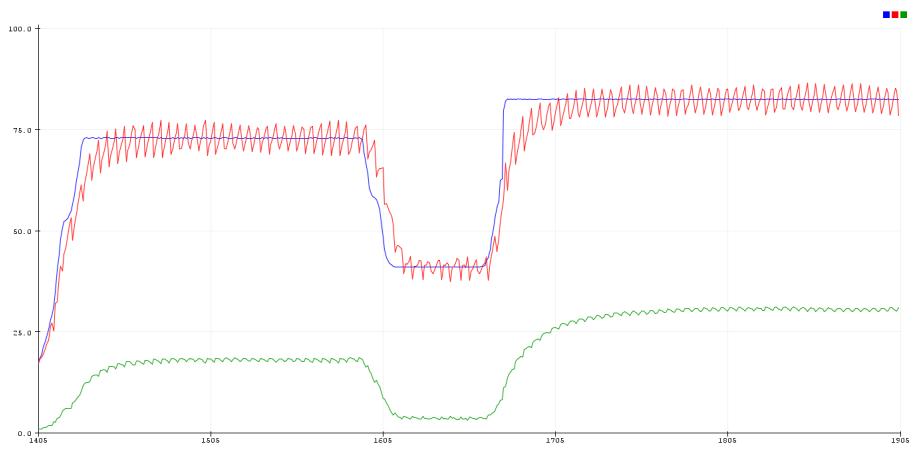
    Serial.print(r);
    Serial.print(" u");
    Serial.print(u);
    Serial.print(" u");
    Serial.println(y); } // koniec funkcie
```

Výsledky regulácie z pohľadu užívateľa môžete vidieť na Obr.5.6 na strane 45 a na Obr.5.7 na strane 46. Modrá čiara predstavuje referenčnú hodnotu, červená čiara predstavuje hodnoty senzora a zelená čiara predstavuje akčný zásah. Na Obr. 5.8 na strane 46 je zobrazený priebeh PID regulácie na Optoshiede v programe MATLAB.

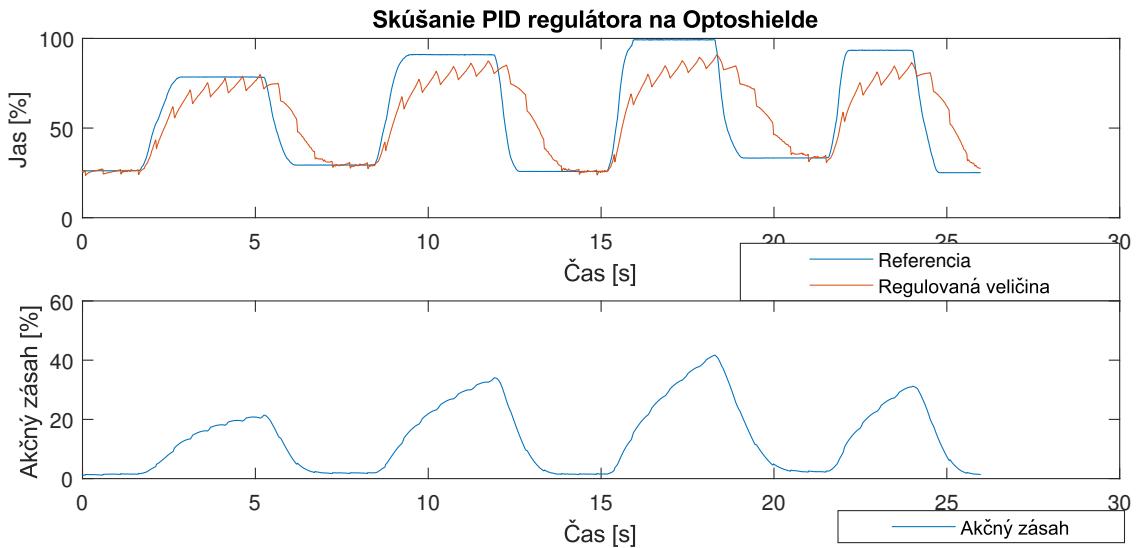
Zdrojový kód príkladu nájdete medzi prílohami, ako prílohu C.



Obr. 5.6: Grafický výstup z PID 1



Obr. 5.7: Grafický výstup z PID 2



Obr. 5.8: Priebeh PID regulácie zobrazený v programe MATLAB

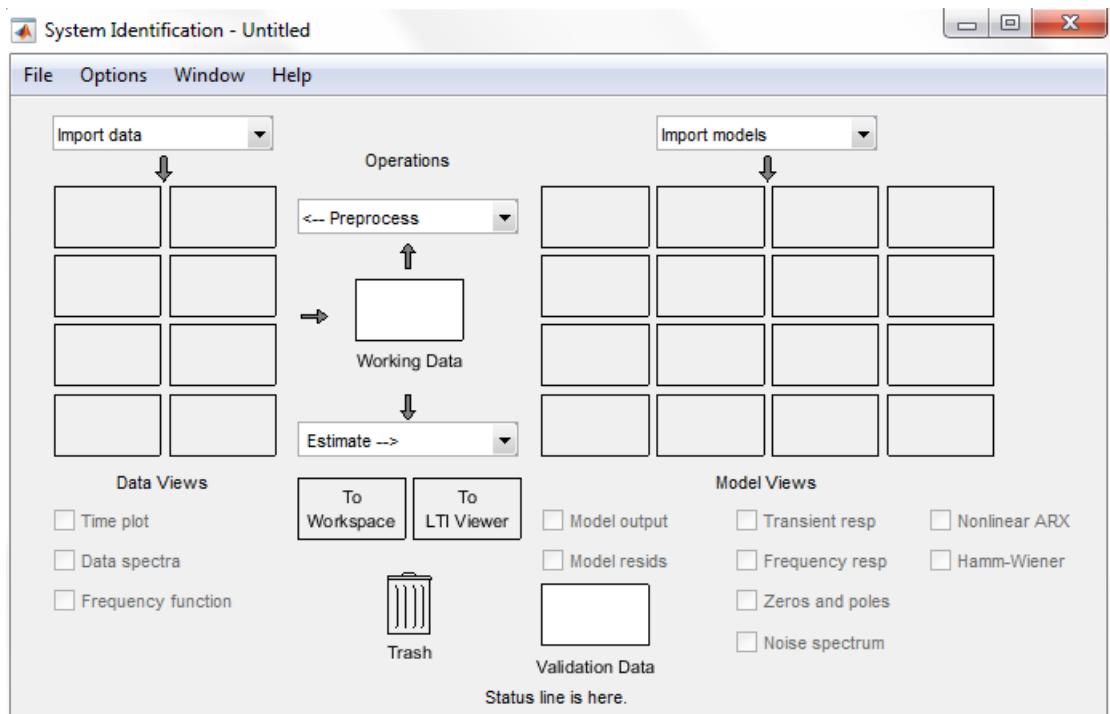
5.4 Identifikácia Optoshieldu

Systémy alebo časti systémov v oblasti automatizácie môžeme popísať matematickými rovnicami, ktoré charakterizujú správanie sa systému. Môže sa stať ale, že nepoznáme matematický model systému (v našom prípade Optoshield), s ktorým pracujeme. Na tento problém ponúka riešenie identifikácia systému.

Identifikácia systému je proces, počas ktorého, podľa vstupno-výstupných údajov meraných na systéme, dostaneme matematický model (prechodovú funkciu) daného systému. Na identifikáciu používame softwareové prostredie MATLAB/Simulink.

V MATLABe zistíme prechodovú funkciu. Používame *identification toolbox*, čo zapneme napísaním príkazu "ident" do "Command Window". Na Obr. 5.8 vidíme "identification toolbox", ktorý sa má objaviť po zadaní príkazu.

V pracovnom prostradí už musia byť importované vstupné a výstupné dátá namerané na systéme. Pri Optoshielde som používal dátá zo skokovej funkcie, keď Setpoint bol



Obr. 5.9: Identification toolbox v programe MATLAB

100%. Dáta som uložil do súboru .txt pomocou programu CoolTerm. V rolovacom menu "Import data" si vyberieme "Time domain data...". V časti "Workspace Variable" si zadáme premenné ktoré obsahujú namerané vstupné a výstupné dátá systému. Po nastavení počiatočného a vzorkovacieho času klikneme na tlačidlo "Import". Pod rolovacom menu na ľavej časti sa objavia importované dátá. V rolovacom menu "Estimate" si vyberieme "Process models". Vypneme možnosť "Delay", lebo systém je bez oneskorenia a klikneme na tlačidlo "Estimate". Vygeneruje matematický model systému, ktorého prechodová funkcia má zosilnenie K a časovú konštantu $Tp1$. V prípade Optoshieldu zosilnenie $K = 0.99944$ a časová konštanta $Tp1 = 5.9817 \cdot 10^{-5}$. Na Obr. 5.10 na strane 48 sú zobrazené odhadované parametre matematického modelu.

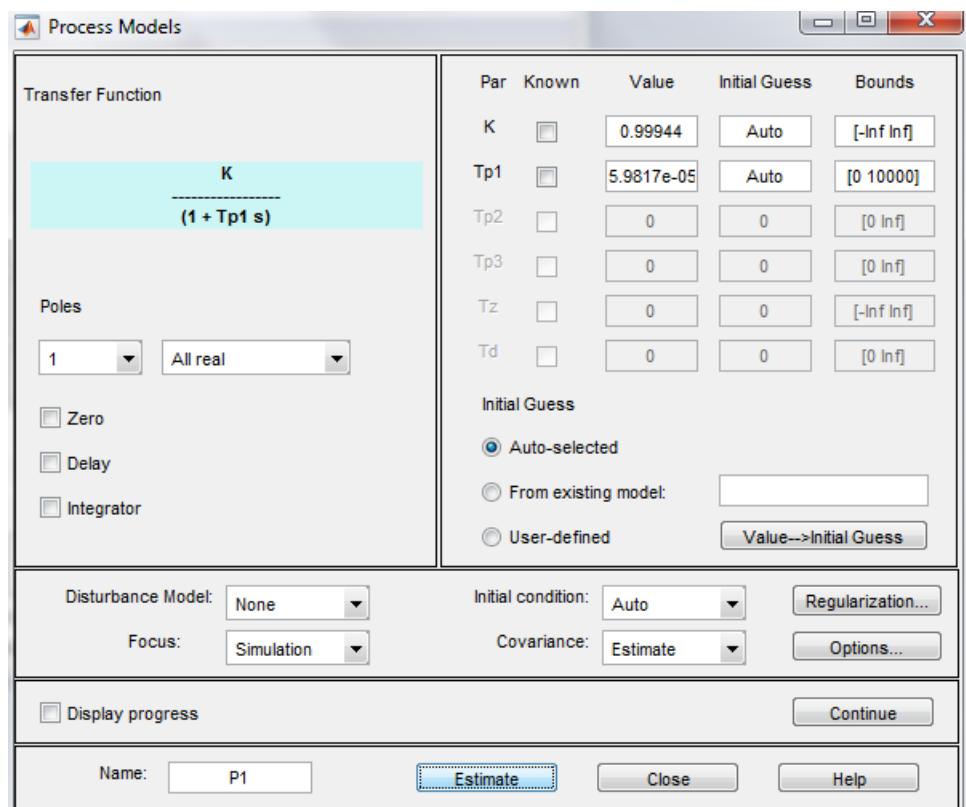
Po kliknutí na tlačidlo "Model output" sa zobrazí náš model a percentá, do akéj miery sú pokryté merané dátá simulovanými. V našom prípade pokrytie bolo 93.99%, čo môžeme považovať za dobré. Merané a simulované dátá sú zobrazené na Obr. 5.11 na strane 48.

Zadaním príkazu $sys = tf([0.99944], [0.0000598171])$ do pracovného prostredia MATLAB vytvoríme spojitú prechodovú funkciu modelu [21]. Spojitá prechodová funkcia vyzera nasledovne, vid' Rov. 5.3.

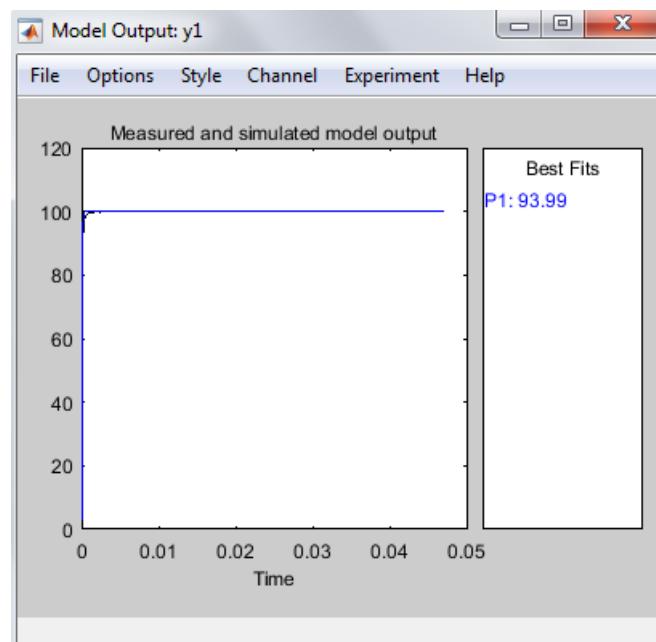
$$tf = \frac{0.99944}{0.00005982s + 1}. \quad (5.3)$$

Prechodová funkcia je v spojitom čase a musíme ju diskretizovať príkazom $c2d(sys, 0.0001, 'zoh')$ [22]. Prvý parameter sys je názov spojitej prechodovej funkcie, potom 0.0001 je vzorkovací čas v sekundách (0.1ms) a posledný parameter ' zoh' ⁵ je tvarovač nultého rádu. Diskrétna

⁵ZOH - Zero Order Hold



Obr. 5.10: Process Model s vygenerovanými konštantami

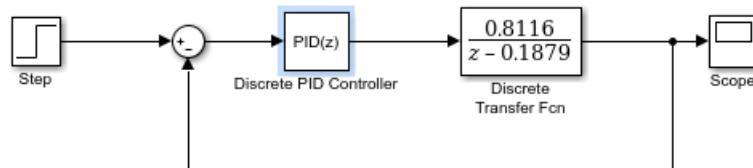


Obr. 5.11: Zobrazené merané a simulované dáta

prechodová funkcia Optoshieldu je nasledovná, vid' Rov. 5.4.

$$tf = \frac{0.8116}{z - 0.1879}. \quad (5.4)$$

Otvoríme program simulink zadaním príkazu "simulink" do pracovného prostredia MATLAB. Zostavíme model regulačného obvodu zobrazený na Obr. 5.12 na strane 49.

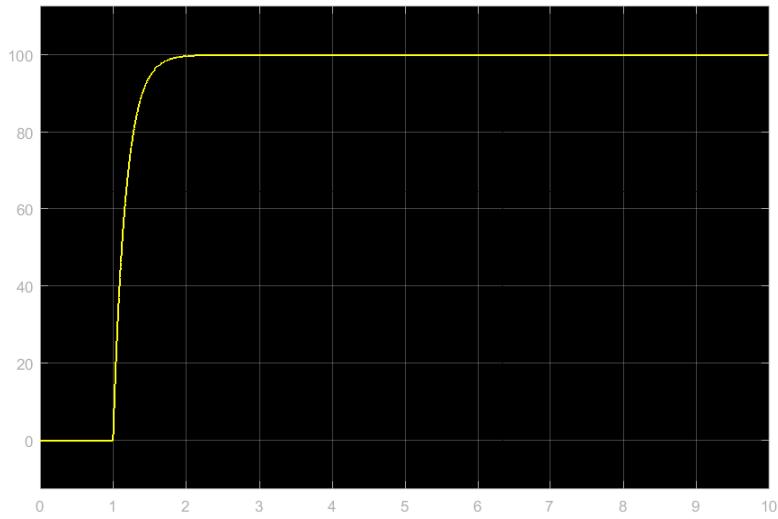


Obr. 5.12: Zostavený model v programe Simulink

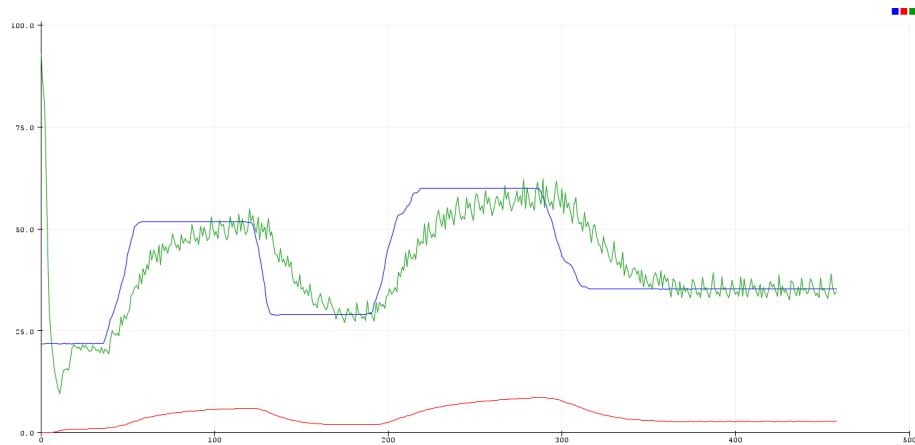
Vstupom do systému je jednotkový skok. Nasleduje sumačný člen ktorý vygeneruje regulačnú odchýlku odčítaním regulovanej veličiny (záporná spätná väzba) zo žiadanej (jednotkový skok). Ako regulátor používame diskrétny PID regulátor. Regulovaná sústava je reprezentovaná diskrétnej prechodovej funkciou do ktorej sme si zadali údaje vygenerované MATLAB-om.

Nastavíme si parametre jednotkového skoku. "Step Time" je čas v ktorom sa má nastať skok, je to 1 s, "Initial value" je 0, "Final value" je 100 a "Sample Time" je 0.0001. Nastavené parametre diskrétnej prechodovej funkcie môžete vidieť na Obr. 5.13 na strane 50. Po kliknutí na PID regulátor nastavíme si vzorkovací čas a klikneme na tlačidlo "Tune...". Nastavíme si tvar prechodovej charakteristiky posúvaním tlačidiel "Response Time" a "Transient behaviour". Ak máme požadovaný tvar klikneme na "Update Block" a vygeneruje parametre PID regulátora, ktoré môžeme používať pri príklade na PID.

Systémom naladené konštanty sú nasledovné: $K_P = 0.00173244161800246$, $K_I = 10.0061553434575$ a $K_D = 0$. Výsledok simulácie v prostredí Simulink je zobrazený na Obr. 5.13 na strane 50. Konštanty aplikujeme v zdrojovom kóde príkladu PID regulácia pomocou Optoshieldu. Aplikované hodnoty z pohľadu užívateľa, v programe Arduino IDE sú zobrazené na Obr. 5.14 na strane 50. Modrá farba reprezentuje referenčnú hodnotu, zelená regulovanú veličinu ktorú červenou farbou je označený akčný zásah systému. Výsledky v programe MATLAB sú zobrazené na Obr. 5.15 na strane 51.



Obr. 5.13: Výsledok simulácie



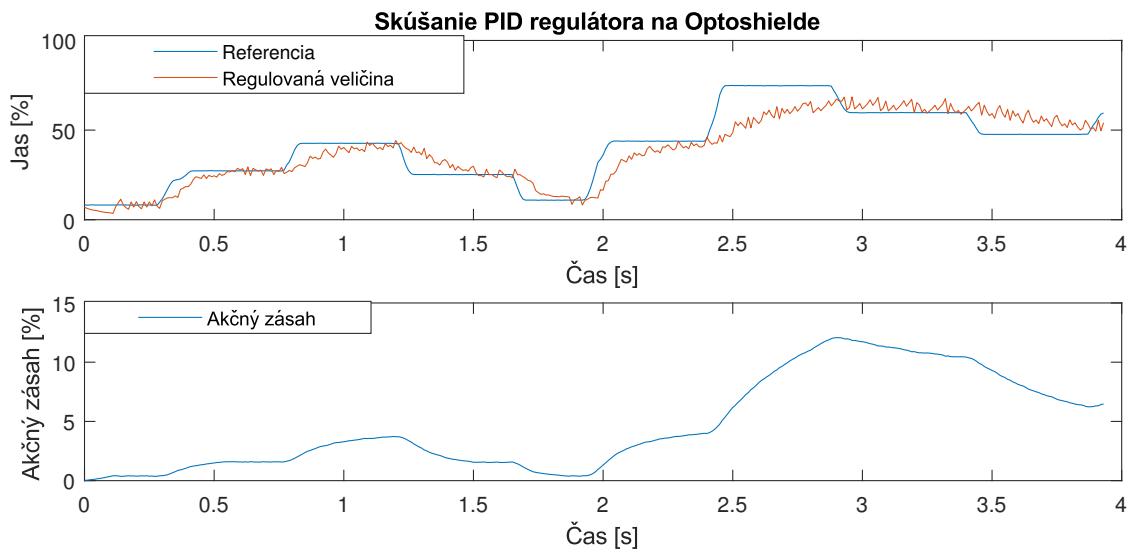
Obr. 5.14: Aplikácia zistených konštánt

5.5 Skoková funkcia pomocou Motoshieldu

Skokový experiment na Motoshielde je realizovaný nastavením rýchlosťi motora a následným sledovaním zmeny otáčok.

Rýchlosť si nastavíme v časti `void setup()` funkciou `MotoShield.setMotorSpeed(Setpoint);`, kde `Setpoint` je premenná typu `float` a predstavuje rýchlosť motora v percentách. V časti `void loop()` zmeráme otáčky funkciou `MotoShield.readRevolutions()` a zmerané hodnoty následne konvertujeme na percentá pomocou funkcie `AutomationShield.mapFloat()`. Hranice prevodu boli pred experimentom zmerané a sú staticky zadané pomocou premenívých `minimum` a `maximum`.

Hraničné hodnoty môžeme overiť aplikáciou príkazu `Serial.println(senzor);`. Minimálnu hodnotu treba zmerať pri rýchlosťi 30% a maximálnu pri rýchlosťi 100%.



Obr. 5.15: Aplikácia zistených konštant

Zdrojový kód 5.4: Funkčná časť skokovej funkcie na Motoshield

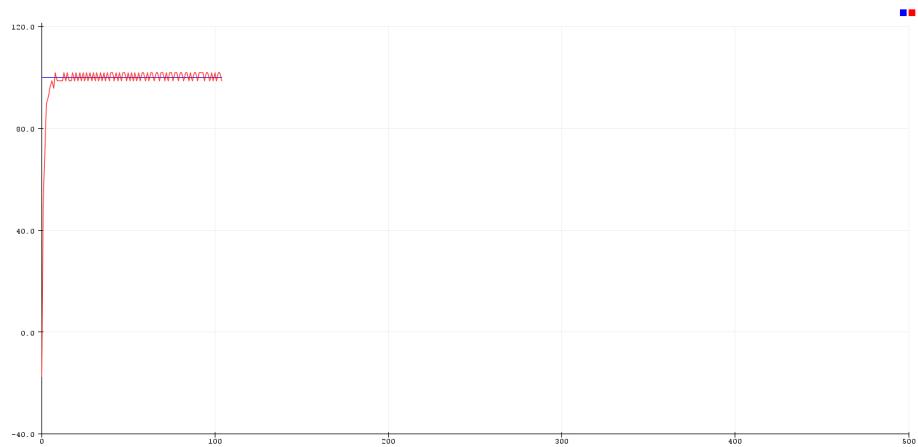
```
void step(){
    senzor = MotoShield.readRevolutions();
    converted = AutomationShield.mapFloat(senzor, minimum, maximum
        , 0.00, 100.00);

    Serial.print(Setpoint);
    Serial.print(", ");
    Serial.println(converted);
}
```

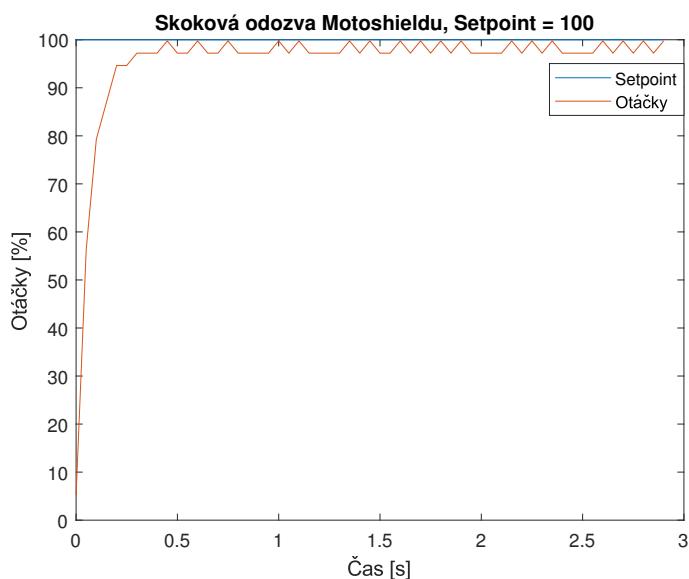
Výsledky testu z hľadiska používateľa môžete vidieť na Obr.5.16. Modrá čiara predstavuje referenčnú rýchlosť a červenou čiarou sú označené prekonvertované otáčky motora do rozsahu 0-100.

Na Obr. 5.17 na strane 52 môžeme vidieť skokovú odozvu Motoshieldu v programe MATLAB.

Zdrojový kód príkladu nájdete medzi prílohami, ako príloha D.



Obr. 5.16: Skoková odozva Motoshieldu, Setpoint = 80



Obr. 5.17: Skoková odozva Motoshieldu v programe MATLAB, Setpoint = 100

5.6 PID regulácia pomocou Motoshieldu

Tento príklad aplikuje PID reguláciu na plošnom spoji Motoshield. Riadi otáčky motora podľa užívateľom zadanej referenčnej hodnoty riadenými akčnými zásahmi. Referenčná hodnota je zadaná pomocou potenciometra. Cieľom príkladu je, aby študent vhodným nastavením parametrov regulátora dosiahol čo nejjemnejší výstupný signál s rýchlosťou reakciou na zmenu referenčnej hodnoty a bez prekmitov. Na časovanie používame funkcie `Sampling()`. Koncepcia príkladu je podobná predchádzajúceho. Zdrojový kód celého príkladu nájdete v prílohách pod názvom C.

Časovanie nastavíme podobne ako v PID príklade pre Optoshield. Pokračujeme v sekcií `void setup()` inicializáciou dosky pomocou funkcie `MotoShield.begin()` a nastavením smeru otáčania funkciou `MotoShield.setDirection(true)`. Nasleduje funkcia `setMotorSpeed(100)` s príkazom `delay(1000)`. Táto dvojica roztáča motor na plné otáčky

na jednu sekundu. Slúži ako kalibrácia, potrebujeme počiatočné hodnoty otáčky aby sme mohli v ďalšej vypočítať regulačnú odchýlku, čo je rozdielom referencie a otáčok. Sekcia `void setup()` ešte obsahuje funkcia na nastavenie parametrov PID regulátora.

Zdrojový kód 5.5: Zdrojový kód príkladu PID - časť `setup()`

```
void setup() {  
  
    Serial.begin(9600);  
  
    MotoShield.begin();           // inicializacia dosky  
    MotoShield.setDirection(true); // nastavenie smeru otacania  
  
    MotoShield.setMotorSpeed(100); // kalibracia  
    delay(1000);  
  
    Sampling.interruptInitialize(Ts * 1000);    // nastavenie  
                                                // casovania  
    Sampling.setInterruptCallback(stepEnable); // nastavenie ISR  
  
    // nastavenie konstant pre PID  
    PIDAbs.setKp(0.007);  
    PIDAbs.setTi(0.015);  
    PIDAbs.setTd(0.0002);  
} // koniec funkcie setup()
```

Funkčná časť programu sa nachádza vo funkcií `step()`. Referenčnú hodnotu čítame funkciou `MotoShield.referenceRead()` a otáčky motora v percentách funkciou `MotoShield.readRevolutionsPerc()`. Regulačná odchýlka je uložená v premennej `error` a je rozdielom referencie a výstupu zo senzora (otáčok v percentách). Akčný zásah vypočítame rovnaku ako pri Optoshield, funkciou `u = PIDAbs.compute(error, 0, 100, 0, 100)`; a aplikujeme ho funkciou `MotoShield.setMotorSpeed(u)`.

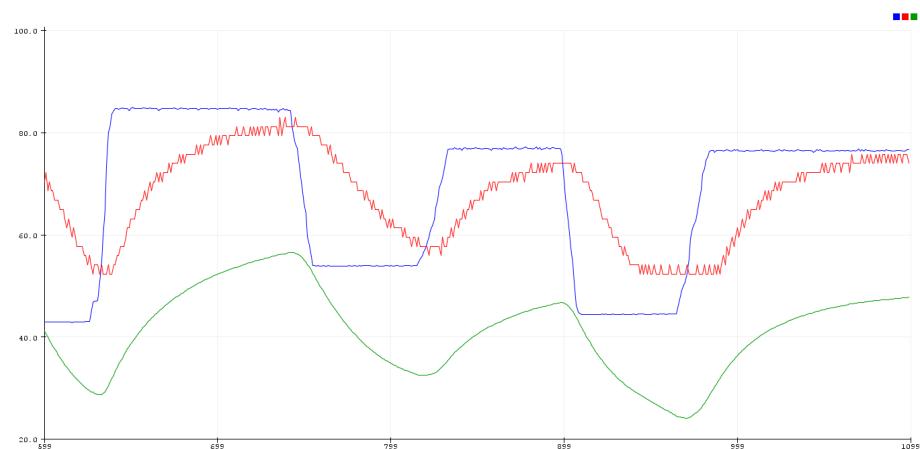
Zdrojový kód 5.6: Zdrojový kód príkladu PID - časť `step()`

```
void step(){  
  
    r = MotoShield.referenceRead();           // citanie referencie  
    y = MotoShield.readRevolutionsPerc(); // citanie otacok  
  
    error = r - y;                         // regulacna odchylka  
  
    u = PIDAbs.compute(error, 0, 100, 0, 100) // citanie kacneho  
                                                // zasahu;  
  
    MotoShield.setMotorSpeed(u);             // aplikacia akcneho  
                                                // zasahu  
  
    Serial.print(r);
```

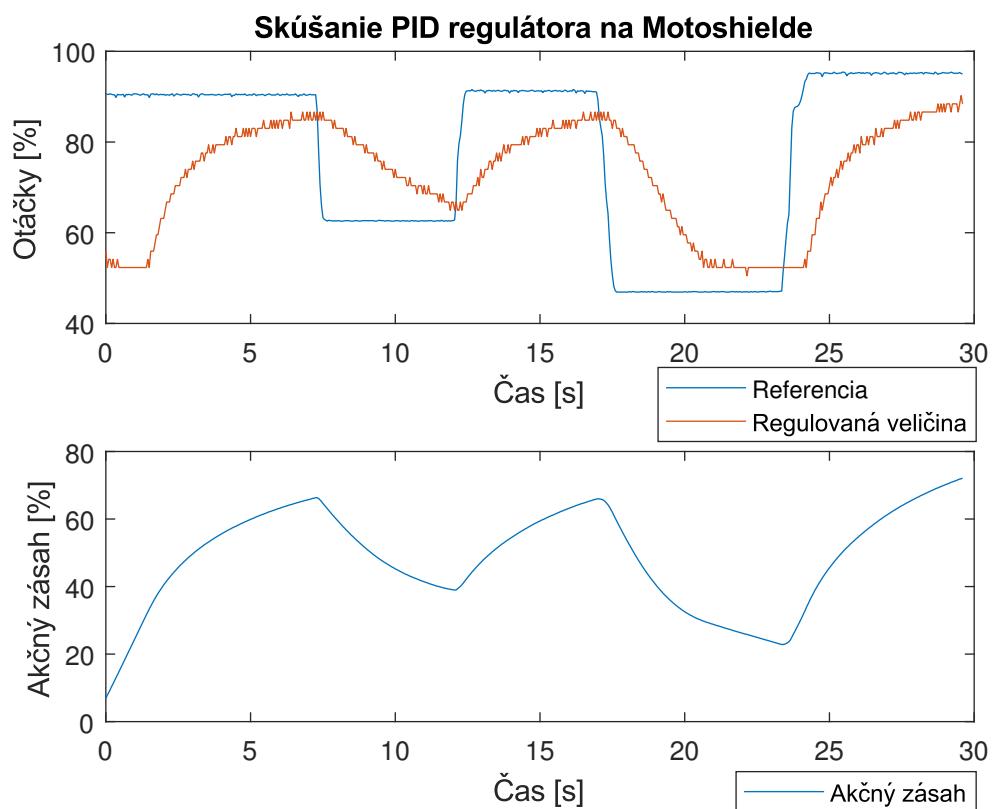
```

Serial.print("█");
Serial.print(y);
Serial.print("█");
Serial.println(u);
} // koniec funkcie step()

```



Obr. 5.18: Grafický výstup z PID 1



Obr. 5.19: Priebeh PID zobrazený v programe MATLAB

Výsledky regulácie z pohľadu užívateľa môžete vidieť na Obr. 5.18. Modrá čiara predstavuje referenciu, červená čiara predstavuje otáčky motora v percentách a zelená čiara predstavuje rideň akčný zásah motora. Zobrazené výsledky v programe MATLAB môžete vidieť na Obr.5.19. Z výsledkov vyplýva, že regulovaná veličina sa rozkmitá pri náhľej zmene referencie. Môžeme dosiahnuť presnejší výsledok zmenou konštant regulátora, napríklad znížením proporcionálnej konštanty K_P , aby sme zjednili mieru zmeny reguloanej veličiny. Presné konštanty môžeme získať identifikáciou sústavy pomocou programu MATLAB/SIMULINK.

Zdrojový kód príkladu sa nachádza medzi prílohami, ako príloha E.

5.7 Identifikácia Motoshieldu

Motoshield identifikujeme podobne, ako Optoshield. Náš cieľ je rovnaký, hľadáme prechodovú funkciu systému a pomocou prechodovej funkcie v programe Simulink určíme parametre PID regulátora. Na identifikáciu Motoshieldu som používal údaje merané pri aplikácii jednotkového skoku. Pri meraní použitý zdrojový kód sa nachádza medzi prílohami, ako príloha F.

Začíname v MATLAB-e, importovaním vstupno-výstupných dát. Vstup do systému bola rýchlosť motora, ako PWM signál, čiže hodnota je z intervalu 0-255. Použil som maximálnu hodnotu. Výstupom zo systému boli otáčky v rad/s.

Prevod otáčok na rad/s je nasledovný, viď. Rov. 5.5:

$$rad/s = ot/min * \frac{2\pi}{60}. \quad (5.5)$$

Otáčky som počítal nasledovne: Pomocou funkcie `Sampling()` som sledoval hodnotu premennej `pulse`, ktorej hodnota sa zvýši o jednu, ak hallova sonda detekuje severný pól magneta. Vzorkovací čas bol 50 ms. Získanú hodnotu som vynásobil najprv 20-timi (dostali sme hodnotu za jednu sekundu), potom 60-timi (hodnota za minutu), to je spolu 1200. Toto číslo následne vydelíme 2660-timi (2660 predstavuje jednu otáčku predného hriadeľa motora) a výsledná hodnota bude v otáčkach za minutu. Na konci vynulujem hodnotu premennej `pulse` sme a povolíme prerušenia príkazom `interrupts()`, lebo na začiatku boli zakázané príkazom `noInterrupts()`.

Zdrojový kód 5.7: Program na počítanie otáčok - časť step()

```
void step(){
noInterrupts();

rev = (float(pulse) * 1200.00) / 2660.00; // rev/min

omega = (rev * (2 * Pi)) / 60; // rad/s

pulse = 0;

Serial.print(rev);
Serial.print(",");
```

```

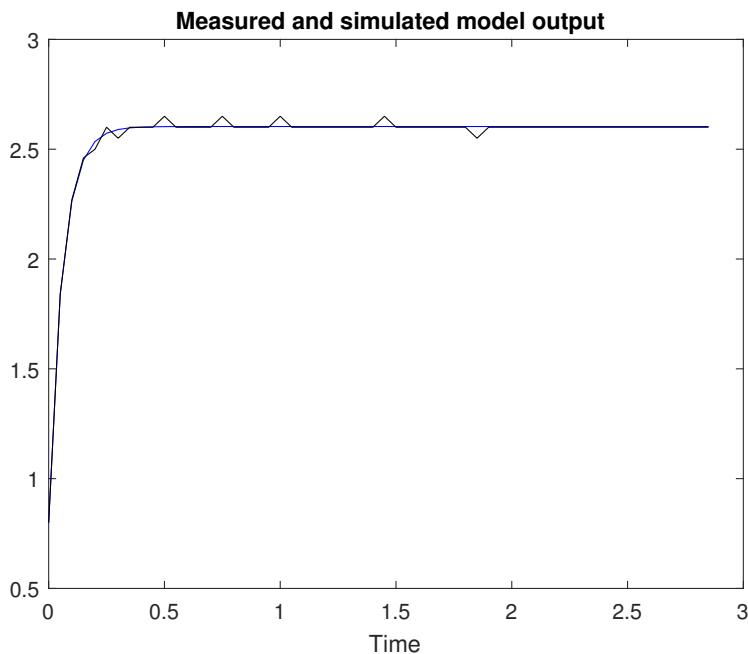
Serial.println(omega);
interrupts();
}

```

Ak dátá sú importované, zapneme identifikačný toolbox príkazom `ident`. Podobne importujeme vstupno-výstupné dátá do identifikačného toolboxu vybratím "Import data" → "Time domain data...". Klikneme na "Estimate" → "Transfer Function Models...". Vyberieme si model s dvomi póľmi a bez núl. Systém vygeneruje prechodovú funkciu, ktorá v našom prípade je znázornená na Rov. 5.6.

$$tf = \frac{12.43}{s^2 + 92s + 1218}. \quad (5.6)$$

Model pokrýva 93.57 % grafu, ako je zobrazený na Obr. 5.20. Čierna krivka predstavuje merané dátá a modrá krivka predstavuje náš model.



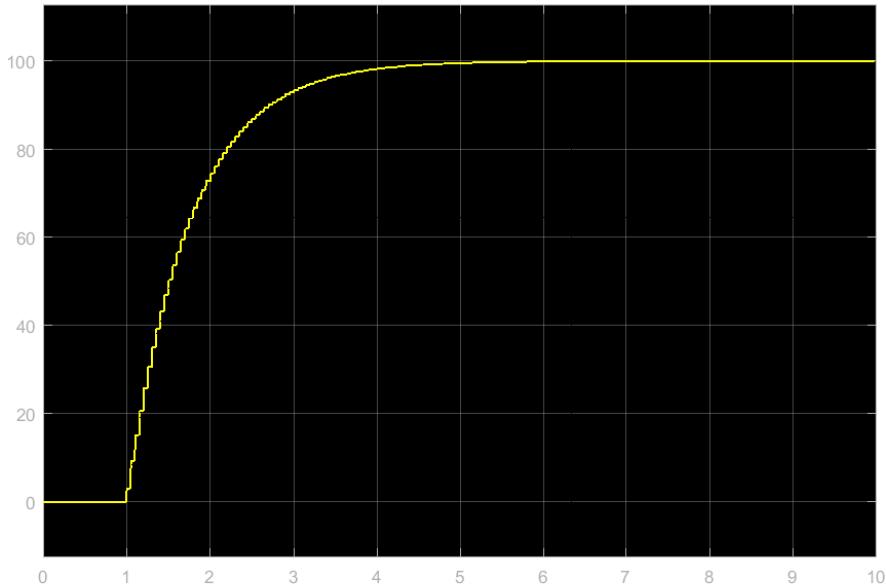
Obr. 5.20: Pokrytie meraných dát modelom

Spojité prechodovú funkciu zmeníme na diskrétnu pomocou príkazu `c2d (sys, 0.05, 'zoh')`. Prvá vstupná hodnota `sys` predstavuje spojité prechodovú funkciu, potom nasleduje vzorkovací čas v sekundách a tvarovač nultého rádu. Diskretizovaná funkcia je zobrazená na Rov. 5.7.

$$tf = \frac{0.004464z + 0.001037}{z^2 - 0.471z + 0.01005}. \quad (5.7)$$

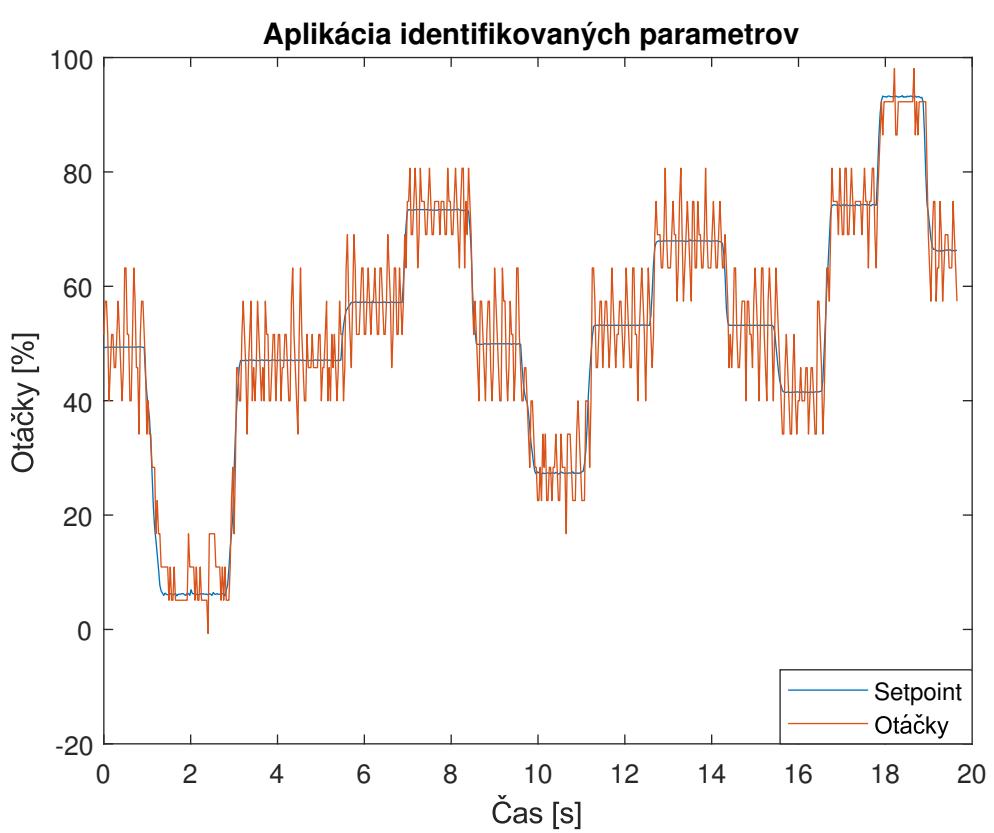
Po vygenerovaní diskrétnej prechodovej funkcie zapneme Simulink a zostavíme si regulačný obvod, podobne ako v prípade Optoshieldu. Po nastavení vstupu a po zadaní diskrétnych parametrov systému môžeme ladiť regulátor. Po ladení spustíme simuláciu a ak

sme spokojní s výsledkom môžeme aplikovať vygenerované konštanty. Simulinkom vygenerované parametre v prípade Motoshieldu sú nasledovné: $K_P = 3.07348889703209$, $K_I = 122.939555881283$ a $K_D = 0$. Výsledok simulácie v programe Simulink je zobrazený na Obr. 5.21.



Obr. 5.21: Výsledok simulácie

Aplikované parametre regulátora sú zobrazené na Obr. 5.22. Ako môžeme vidieť otáčky sú premenlivé pri stálej referencie. Tieto odchýlky môžu byť spôsobené nepresným modelom (s presnejším modelom by sme mohli mať presnejšie výsledky), zlým nastavením regulátora v Simulinku (iný tvar prechodovej charakteristiky môže generovať iné konštanty) aleb aj vzorkovacím časom (častejšie vzorkovanie môže generovať plynulejší graf). Ako vidíme, tento graf tvarovo sa líši od grafu znázorneného na Obr.5.19 na strane 54. Je to spôsobené tým, že tu bola použitá iná funkcia na počítanie otáčok. Novú metódu si môžete nájsť v prílohe F.



Obr. 5.22: Aplikované parametre regulátora

6 Záver

Cieľom bakalárskej práce bol návrh experimentálnych modulov vhodných na použitie vo výučbe. V rámci práce boli navrhnuté dva moduly, Optoshield a Motoshield.

Prvá kapitola je úvodom do problematiky. Druhá kapitola sa zaoberá s našou motiváciou, čo nás viedlo k návrhu vlastných učebných pomôcok.

V tretej kapitole je opísaný proces návrhu Optoshieldu. Kapitola má dve časti, hardwareovú a softwareovú. Hardwareová časť začína s charakterizáciou použitých súčiastok. Čitateľ musí pochopiť na akom princípe funguje elektronický obvod, s ktorým pracuje. Potom jednoduchšie pochopí návrh plošného spoja a funkcie napísané pre daný modul. Kapitola pokračuje s návrhom schémy a plošného spoja v počítačovom programe Dip-Trace. V prvej kapitole je vysvetlený proces návrhu a implementácie vlastnej súčiastky, lebo nie všetky nami použité súčiastky sa nachádzajú v databáze programu. Popísané veci sú zobrazené na obrázkach, aby pochopenie vysvetlených postupov bolo ešte ľahšie. Softwareová časť sa začína s krátkym úvodom o programovačom jazyku Arduino a o štruktúre knižníc použitých pre hardware kompatibilného s Arduinom. Je vysvetlený aj proces vytvorenia vlastnej knižnice. Vedieť si vytvoriť vlastnú knižnicu môže byť dôležitý v budúcnosti, ak človek si navrhne vlastný hardware a chce si uľahčiť prácu s ním. Softwareová časť ďalej obsahuje popis funkcií napísaných pre modul. Popis obsahuje typ funkcie, typ hodnôt vrátených funkciou (v prípade, ak funkcia vráti nejaké hodnoty), na čo slúži funkcia a na akom princípe pracuje. Na konci je priložený a okomentovaný zdrojový kód.

Tretia kapitola obsahuje proces návrhu Motoshieldu. Štruktúra kapitoly je rovnaká ako v prípade tretej kapitoly o Optoshielde. Začína sa s charakterizáciou použitých súčiastok, nasleduje návrh schémy a plošného spoja a kapitola sa skončí s funkciami napísanými pre modul.

Piata kapitola je dôležitá kapitola, zaoberá sa s didaktickými príkladmi. V rámci príkladoch sú aplikované princípy z oblasti automatizácie ako skoková funkcia, PID regulácia alebo identifikácia dosky. Každý príklad je rozpracovaný do samostatnej sekcie. Na začiatku sekcie je popísaný na čo je príklad zameraný a čitateľ sa dozvie, že na čo sa používa v príklade aplikovaný princíp. V niektorých prípadoch sa v texte nachádzajú aj schémy, ktoré pomôžu čitateľovi lepšie pochopiť danú látku. V príklade je uvedený, pomocou ktorých súčiastok je daný princíp aplikovaný a sú vysvetlené dôležité časti zdrojového kódu. Vzhľadom na dĺžku, zdrojový kód príkladov, ako celok je uvedený ako príloha. Na konci každého príkladu sú grafické výsledky experimentov. Sú zobrazené výsledky z Arduino IDE, lebo tieto slúžia užívateľovi ako prvotná spätná väzba a nachádzajú sa tam aj grafy zobrazené v programe MATLAB, ktoré umožňujú podrobnejšiu analýzu.

Príklady zároveň slúžia na overenie vhodnosti modulov na použitie vo výučbe. Počas skúšania sa nám podarilo odhadnúť viacero chýb - aj softwareových aj hardwareových.

Napríklad, jedna z chýb Optosheldu boli nevhodne zvolené hodnoty odporov R1-R4, ktoré pôsobili nepoužiteľné výsledky meraní. Ďalšia hardwareová chyba Optoshieldu je, že pri kalibrácii (podla svetelných podmienok) sa vonkajšie svetlo cez dno svetelnej diódy dostane do vnútra trubice a pokazí kalibráciu. Nevhodne nakalibrovaná doska produkuje nepresné, mätúce výsledky. Túto chybu by sa dalo eliminovať napríklad natretím dna diódy tmavou farbou (overené riešenie). Ako návrh do budúcnosti by som vymenil klasické rezistory na SMD aby som mohol ušetriť nejaké miesto a aplikoval by som ďalší pársvetelnnej diódy a fotorezistora v trubici. Svetelná dióda by mohla byť inej farby, napr. červená alebo žltá. Ďaľšie možné riešenie je pripojenie svetelnej diódy na samostatný pin, aby sme mohli regulovať ich jas osobitne. Optoshield by som navrhoval takým ľuďom, ktorí prvý krát pracujú s Arduinom. , Napriek jeho nedostatkom, je všeobecná a použiteľná učebná pomôcka. Umožňuje vyskúšanie základných funkcií Arduina a vďaka vstavanej voľnej svetelnej diódy môžu vykonať malé experimenty a napísat si prvé vlastné kódy. Ak časom pokročia, môžu pokračovať s napísanými didaktickými príkladmi.

Na testovanie Motoshieldu som mal menej času. Jeho softvér a funkcie napísané pre Motoshield sa neustále vyvýjajú a zlepšujú. Zlepšenie hardvéru je možné tiež. Ďalšie testovanie a skúšanie obvodu s operačným zosilňovačom je potrebné, aby sme vedeli posunúť jeho pracovné hraničné hodnoty. V procese návrhu som experimentoval s prúdovým senzorom ACS 712, ktorý sa neosvedčil a bol nahradený zosilňovačom, ale nevylúčim testovanie ďalších senzorov. Uvažoval by som nad pridávaním pasívnych filtrov na zjemnenie signálu pri meraní prúdu alebo napäťia. Aj rad didaktických príkladov je rozšíriteľný.

Nápady na zlepšenia a prípadné, ešte skryté chyby nám ponúkajú ďaľšie výzvy do budúcnosti, ktoré môžu tvoriť tému pre ďalšie bakalárské práce alebo skupinové projekty.

A Zdrojový kód funkcie Sampling()

Zdrojový kód A.1: Zdrojový kód funkcie Sampling()

```
void Sampling(int sampleTime){

    Ts = sampleTime;

    float Lt = 65535;           // maximalna mozna hodnota
                                // registra

    float Tr1 = 0.0625;         // resolution at p = 1
    float Tr8 = 0.5;            // resolution at p = 8
    float Tr64 = 4;             // resolution at p = 64
    float Tr256 = 16;           // resolution at p = 256
    float Tr1024 = 64;          // resolution at p = 1024

    float resolution[] = {Tr1 , Tr8 , Tr64 , Tr256 ,
                          Tr1024};                         // pole obsahujuce rozlisenia

    float uTs = (float(sampleTime) * 1000.00);
                // vzorkovaci cas v
                // mikrosekundach

    // cyklus pomoze pri vybere vhodneho preskalovaca
    for(_i = 0; _i < 5; _i++){
        _testValue = uTs / resolution[_i];
        if(_testValue <= Lt){
            break;
        } // koniec podmienky
    } // koniec cyklu

    // ***** Nastavenie bitov registra Timer1 *****
    TCCR1A = 0;                  // resetuje vsetky byty
    TCCR1B = 0;
    TCNT1  = 0;
```

```

cli();           // prerusenia nie su
                // povolene

OCR1A = _testValue - 1;    // max. hodnota pre reset

TCCR1B |= (1 << WGM12); // CTC mode

// volba preskalovaca
switch (_i){

case 0:
TCCR1B |= (1 << CS10); // preskalovac 1 (zaklad)

TCCR1B |= (0 << CS11); // ostatne bity nastavim ako
0
TCCR1B |= (0 << CS12);
break;

case 1:
TCCR1B |= (1 << CS11); // prescaler 8

TCCR1B |= (0 << CS10);
TCCR1B |= (0 << CS12);
break;

case 2:
TCCR1B |= (1 << CS10); // prescaler 64
TCCR1B |= (1 << CS11);

TCCR1B |= (0 << CS12);
break;

case 3:
TCCR1B |= (1 << CS12); // prescaler 256

TCCR1B |= (0 << CS10);
TCCR1B |= (0 << CS11);
break;

case 4:
TCCR1B |= (1 << CS10); // prescaler 1024
TCCR1B |= (1 << CS12);

TCCR1B |= (0 << CS11);
break;
}

```

```
default:  
Serial.println("The sampling time is higher than the  
limit, please try smaller value");  
  
}  
  
// po vybrati vhodneho preskalovaca povoli prerusenia  
  
TIMSK1 |= (1 << OCIE1A); // povoli timer prerusenia  
sei(); // povoli globalne  
// prerusenia  
  
}// koniec funkcie
```

B Skoková funkcia pomocou Optoshieldu

Zdrojový kód B.1: Zdrojový kód príkladu na skokovú funkciu

```
#include "AutomationShield.h"

float Setpoint = 70.00;                      // uroven jasu

bool enable=false;
unsigned long int curTime=0;                  // premenne na
                                                // casovania
unsigned long int prevTime=0;

unsigned long Ts = 0.1;                        // vzorkovaci cas

void setup() {

    Serial.begin(115200);

    Sampling.interruptInitialize(Ts * 1000);
                                                // inicializacia
                                                // casovania
    Sampling.setInterruptCallback(stepEnable);

    OptoShield.begin();                         // inicializacia dosky
    OptoShield.calibration();
    delay(1000);
    OptoShield.actuatorWrite(Setpoint);
                                                // nastavenie hodnoty
                                                // jasu
}

void loop() {

    curTime=millis();
    if (enable) {
        step();                                // funkcia step()
        enable=false;
```

```
} // end of the if statement
} // end of the loop

void stepEnable(){
enable=true;
}

void step(){                                // funkcn a cast
                                         // programu
float Senzor = OptoShield.sensorRead();

Serial.print(Setpoint);
Serial.print(",");
Serial.println(Senzor);
}
```

C PID regulácia pomocou Optoshieldu

Zdrojový kód C.1: Zdrojový kód príkladu na PID reguláciu()

```
#include "AutomationShield.h"

unsigned long Ts = 10;           // vzorkovaci cas v
                                // milisekundach

bool enable=false;              // tzv. vlajka, sleduje
                                // hodnotu premennej

float r = 0.00;
float y = 0.00;
float u = 0.00;
float error = 0.00;

void setup() {

    Serial.begin(9600);

    OptoShield.begin();          // inicializacia dosky
    OptoShield.calibration();   // kalibracia

    Sampling.interruptInitialize(Ts * 1000);
                                // inicializovanie sampling
                                // funkcie
    Sampling.setInterruptCallback(stepEnable);
                                // nastavenie funkcie pre
                                // ISR

    // nastavenie konstant pre PID
    PIDAbs.setKp(0.1);
    PIDAbs.setTi(0.015);
    PIDAbs.setTd(0.0000008);

} // end of the setup

void loop() {
```

```

if (enable) {
    step();
    enable=false;
}

} // end of the loop

void stepEnable() {                                // ISR
    enable=true;
}

void step(){                                     // zdrojovy kod ide sem

    r = OptoShield.referenceRead(); // nacitanie referencie
    y = OptoShield.sensorRead();   // nacitanie hodnot
                                    // fotorezistora

    error = r - y;                      // regulacna odchylka

    u = PIDAbs.compute(error,0,100,0,100);
                                    // pocitanie akcneho
                                    // zasahu

    OptoShield.actuatorWrite(u);        // akcny zasah ako jas

    Serial.print(r);
    Serial.print(" ");
    Serial.print(u);
    Serial.print(" ");
    Serial.println(y);
}

```

D Skoková funkcia pomocou Motoshieldu

Zdrojový kód D.1: Zdrojový kód príkladu na skokovú funkciu

```
#include "AutomationShield.h"

float Setpoint = 100.00;           // referencia

bool enable=false;                // pre sampling
unsigned long int curTime=0;
unsigned long int prevTime=0;

unsigned long Ts = 0.1;           // vzorkovaci cas

float senzor = 0.00;             // premenne
float converted = 0.00;

                           // hranice prevodu
float maximum = 24.36;
float minimum = 10.27;

void setup() {

Serial.begin(9600);

Sampling.interruptInitialize(Ts * 1000);
Sampling.setInterruptCallback(stepEnable);
                           // casovanie

MotoShield.begin();              // nastavenie pinov

MotoShield.setDirection(true);   // start motoru
MotoShield.setMotorSpeed(Setpoint);
}

void loop() {

curTime=millis();
```

```
if (enable) {  
    step();  
    enable=false;  
  
} // end of the if statement  
} // end of the loop  
  
void stepEnable(){  
    enable=true;  
}  
  
void step(){  
    senzor = MotoShield.readRevolutions(50);  
                                // meranie otacok  
  
    converted = AutomationShield.mapFloat(senzor,minimum,  
                                          maximum,0.00,100.00);  
                                // prevod otacok  
  
    Serial.print(Setpoint);  
    Serial.print(",");  
    Serial.println(converted);  
}
```

E PID regulácia pomocou Motoshieldu

Zdrojový kód E.1: Zdrojový kód príkladu na PID reguláciu()

```
#include "AutomationShield.h"

unsigned long Ts = 10;                      // vzorkovaci cas v
                                                // milisekundach

bool enable=false;                          // tzv. vlajka, sleduje
                                            // hodnotu premennej

float r = 0.00;
float y = 0.00;
float u = 0.00;
float error = 0.00;

void setup() {
    Serial.begin(9600);

    MotoShield.begin();                     // inicializacia dosky
    MotoShield.setDirection(true);          // nastavenie smeru
                                            // otacania

    MotoShield.setMotorSpeed(100);           // kalibracia
    delay(1000);

    Sampling.interruptInitialize(Ts * 1000); // nastavenie casovania
    Sampling.setInterruptCallback(stepEnable);
                                            // nastavenie ISR

    // nastavenie konstant pre PID
    PIDAbs.setKp(0.007);
    PIDAbs.setTi(0.015);
    PIDAbs.setTd(0.0002);
```

```

}// koniec funkcie setup()

void loop() {

if (enable) {
step();
enable=false;

}

} // koniec funkcie loop()

void stepEnable(){                                // ISR
enable=true;
}

void step(){

r = MotoShield.referenceRead();      // citanie referencie
variable = MotoShield.readRevolutions(50);          // citanie otacok

y = AutomationShield.mapFloat(variable
,0.00,25.00,0.00,100.00);                         // prevod otacok

error = r - y;                                     // regulacna odchylka

u = PIDAbs.compute(error,0,100,0,100);             // pocitanie akcneho /
// zasahu

MotoShield.setMotorSpeed(u);                      // aplikacia akcneho
// zasahu

Serial.print(r);
Serial.print("°");
Serial.print(y);
Serial.print("°");
Serial.println(u);
}

```

F Program na meranie údajov na identifikáciu Motoshieldu

Zdrojový kód F.1: Zdrojový kód na identifikáciu Motoshieldu

```
#include "AutomationShield.h"

bool enable=false;
unsigned long int curTime=0;
unsigned long int prevTime=0;

unsigned long int Ts = 50;           // vzorkovaci cas v ms

static unsigned long pulse;

float rev;
float omega;

int Speed = 100;
int conv;

float convertedSpeed;
float convertedRev;

float Pi = 3.14159265359;

void setup() {

Serial.begin(9600);
Sampling.interruptInitialize(Ts * 1000);
Sampling.setInterruptCallback(stepEnable);

MotoShield.setDirection(true);
attachInterrupt(digitalPinToInterruption(MOTO_C1) , countPulses
    , FALLING);
}

void loop() {
```

```

MotoShield.setMotorSpeed(Speed);

conv = AutomationShield.mapFloat(Speed, 0, 100, 0, 255);

curTime=millis();
if (enable) {
step();
enable=false;

}

void stepEnable(){
enable=true;
}

void step(){
noInterrupts(); // zakaze prerusenia

rev = (float(pulse) * 1200.00) / 2660.00; // rev/min

omega = (rev * (2 * Pi)) / 60; // rad/s

pulse = 0;

Serial.print(rev);
Serial.print(",");
Serial.println(omega);
interrupts(); // povoli prerusenia
}

static void countPulses(){ // ISR
pulse++;
}

```

Literatúra

- [1] Bay Area Circuits. Instand DFM Control. <http://instantdfm.bayareacircuits.com>. [cit. 27.04.2018]
- [2] DFRobot. Micro Metal Geared motor w/Encoder. <https://www.dfrobot.com/product-1437.html>. [cit. 01.05.2018]
- [3] ElectronicsTutorials. Hall Effect Sensor. <https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html>. [cit. 01.05.2018]
- [4] Arduino Reference. attachInterrupt(). <https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>. [cit. 02.05.2018]
- [5] Robot Platform. Dual H-Bridge Motor Driver - L293 IC. http://www.robotplatform.com/howto/L293/motor_driver_1.html. [cit. 02.05.2018]
- [6] Arduino.cc. Writing a Library for Arduino. <https://www.arduino.cc/en/Hacking/LibraryTutorial>. [cit. 04.05.2018]
- [7] Arduino Reference. millis(). <https://www.arduino.cc/reference/en/language/functions/time/millis/>. [cit. 05.05.2018]
- [8] Arduino Reference micros(). <https://www.arduino.cc/reference/en/language/functions/time/micros/>. [cit. 05.05.2018]
- [9] EngBlaze. <http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/> [cit. 06.05.2018]
- [10] RobotShop.com. Arduino 101: Timers and Interrupts. <https://www.robotshop.com/letsmakerobots/arduino-101-timers-and-interrupts>. [cit. 06.05.2018]
- [11] STMicroelectronics. L293D,L293DD PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES. [cit. 06.05.2018]
- [12] Fairchild Semiconductor. LM2904,LM358/LM358A,LM258/LM258A Dual Operational Amplifier. [cit. 07.05.2018]
- [13] cplusplus.com. Arrays. <http://wwwcplusplus.com/doc/tutorial/arrays/>. [cit. 09.05.2018]
- [14] Arduino Playground. Reading Rotary Encoders - Waveform. <https://playground.arduino.cc/Main/RotaryEncoders#Waveform>. [cit. 09.05.2018]

- [15] AutomationShield.com. Common Functions / Mathematics / Map floating point numbers. <https://github.com/gergelytakacs/AutomationShield/wiki/Common-functions>. [cit. 12.05.2018]
- [16] Quanser.com. High Fidelity Linear Cart System. <https://www.quanser.com/products/high-fidelity-linear-cart-system/>. [cit. 13.05.2018]
- [17] Cyril Belavý. Základy automatizácie a merania. Slovenská technická univerzita v Bratislave, 2012. Strany 63-66. [cit. 14.05.2018]
- [18] M. Bakošová a M. Fikar. Spätnoväzbové riadenie procesov - Regulátory. www.kirp.ctf.stuba.sk/moodle/mod/resource/view.php?id=19473. [cit. 14.05.2018]
- [19] tutorialspoint.com. Control Systems - Time Response Analysis. https://www.tutorialspoint.com/control_systems/control_systems_time_response_analysis.htm. [cit. 15.05.2018]
- [20] tutorialspoint.com. Response of the First Order System. https://www.tutorialspoint.com/control_systems/control_systems_response_first_order.htm. [cit. 15.05.2018]
- [21] MathWorks. Create transfer function model, convert to transfer function model. <https://www.mathworks.com/help/control/ref/tf.html>. [cit. 21.05.2018]
- [22] MathWorks. Convert model from continuous to discrete time. <https://www.mathworks.com/help/control/ref/c2d.html>. [cit. 21.05.2018]